**RESEARCH PAPER: REGULAR EXPRESSION**

**By Sajith Vellappillil**

**INTRODUCTION**

Regular expressions (called REs, or regexes, or regex patterns) 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.

The regular expression language is relatively small and restricted, so not all possible string processing tasks can be done using regular expressions. There are also tasks that can be done with regular expressions, but the expressions turn out to be very complicated. In these cases, you may be better off writing Python code to do the processing; while Python code will be slower than an elaborate regular expression, it will also probably be more understandable.

**IMPORTANT FUNCTIONS**

• Search Function

The re.search() method takes a regular expression pattern and a string and searches for that pattern within the string. If the search is successful, search() returns a match object or None otherwise. Therefore, the search is usually immediately followed by an if-statement to test if the search succeeded, as shown in the following example which searches for the pattern 'word:' followed by a 3 letter word (details below):

• Match Function

re.match() function of re in Python will search the regular expression pattern and return the first occurrence. The Python RegEx Match method checks for a match only at the beginning of the string. So, if a match is found in the first line, it returns the match object

Matching Versus Searching

Python offers two different primitive operations based on regular expressions: match checks for a match only at the beginning of the string, while search checks for a match anywhere in the string

**Other RegEx Functions**

The re module offers a set of functions that allows us to search a string for a match:

Function Description

findall Returns a list containing all matches

[sub](https://www.w3schools.com/python/python_regex.asp#sub) Replaces one or many matches with a string

[split](https://www.w3schools.com/python/python_regex.asp#split) Returns a list where the string has been split at each match

**Metacharacters**

|  |  |  |
| --- | --- | --- |
| Character | Description | Example |
| [] | A set of characters | "[a-m]" |
| \ | Signals a special sequence (can also be used to escape special characters) | "\d" |
| . | Any character (except newline character) | "he..o" |
| ^ | Starts with | "^hello" |
| $ | Ends with | "world$" |
| \* | Zero or more occurrences | "aix\*" |
| + | One or more occurrences | "aix+" |
| {} | Exactly the specified number of occurrences | "al{2}" |
| | | Either or | "falls|stays" |
| () | Capture and group |  |

Metacharacters are characters with a special meaning:

**Special Sequences**

A special sequence is a \ followed by one of the characters in the list below, and has a special meaning:

|  |  |  |
| --- | --- | --- |
| Character | Description | Example |
| \A | Returns a match if the specified characters are at the beginning of the string | "\AThe" |
| \b | Returns a match where the specified characters are at the beginning or at the end of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string") | r"\bain" r"ain\b" |
| \B | Returns a match where the specified characters are present, but NOT at the beginning (or at the end) of a word (the "r" in the beginning is making sure that the string is being treated as a "raw string") | r"\Bain" r"ain\B" |
| \d | Returns a match where the string contains digits (numbers from 0-9) | "\d" |
| \D | Returns a match where the string DOES NOT contain digits | "\D" |
| \s | Returns a match where the string contains a white space character | "\s" |
| \S | Returns a match where the string DOES NOT contain a white space character | "\S" |
| \w | Returns a match where the string contains any word characters (characters from a to Z, digits from 0-9, and the underscore \_ character) | "\w" |
| \W | Returns a match where the string DOES NOT contain any word characters | "\W" |
| \Z | Returns a match if the specified characters are at the end of the string | "Spain\Z" |

**Sets**

A set is a set of characters inside a pair of square brackets [] with a special meaning:

|  |  |
| --- | --- |
| Set | Description |
| [arn] | Returns a match where one of the specified characters (a, r, or n) are present |
| [a-n] | Returns a match for any lower-case character, alphabetically between a and n |
| [^arn] | Returns a match for any character EXCEPT a, r, and n |
| [0123] | Returns a match where any of the specified digits (0, 1, 2, or 3) are present |
| [0-9] | Returns a match for any digit between 0 and 9 |
| [0-5][0-9] | Returns a match for any two-digit numbers from 00 and 59 |
| [a-zA-Z] | Returns a match for any character alphabetically between a and z, lower case OR upper case |
| [+] | In sets, +, \*, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string |

References:

https://www.w3schools.com/python/python\_regex.asp

<https://www.tutorialspoint.com/python/python_reg_expressions.htm>

https://docs.python.org/3/howto/regex.html

https://www.guru99.com/python-regular-expressions-complete-tutorial.html