# References/Research

## Regular Expressions

### <https://en.wikipedia.org/wiki/Regular_expression>

* RE is a sequence of characters that define a search pattern

<https://github.com/ianmcloughlin/slides-regular-expressions/raw/master/slides.pdf>

* RE are strings that represent patterns of text
* Brackets are used to group the characters together
* They are used to search other strings for patterns

**Special Characters**

|  |  |  |
| --- | --- | --- |
| **.** | To concatenate | a.b means a followed by b |
| **|** | Means or | a|b means a or b |
| **\*** | Zero or more times | a\* means 0 or more a’s |

**Precedence**

* \* comes first
* . after \* but before |
* | comes last
* Treat brackets as individual charcters

### <https://docs.python.org/2/library/re.html>

* Regular expressions can be concatenated to form new regular expressions; if A and B are both regular expressions, then AB is also a regular expression. In general, if a string p matches A and another string q matches B, the string pq will match AB.

This logic can be used in the project to build small NFA’s for the regular expression

**Characters that have a special meaning when using the re library**

|  |  |
| --- | --- |
| **Dot (.)** | Any character except a newline |
| **Caret (^)** | The start of a string |
| **Dollar ($)** | The end of a string |
| **Asterisk (\*)** | Match 0 or more repetitions as possible of the RE |
| **Plus (+)** | Match 1 or more repetitions as possible of the RE |
| **Question (?)** | Match 1 or 0 or more repetitions as possible of the RE |
| **Square Brackets ([])** | A set of characters |
| **Pipe (|)** | Either or |
| **Back slash (\)** | Special sequence |
| **Round Brackets ()** | A group |

**Functions used in the re library**

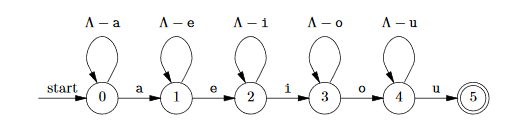
|  |  |
| --- | --- |
| **findall** | Returns list of all matches |
| **search** | Returns an object that if there’s a match in the string |
| **split** | Returns list where the string in split at each match |
| **sub** | Replaces one or more matches with a string |

I will keep these symbols and functions in mind when building my program

## Finite Automata

### <http://infolab.stanford.edu/~ullman/focs/ch10.pdf>

* The finite automaton is a graph-based way of specifying patterns. These come in two varieties, deterministic automata and nondeterministic automata
* Regular expressions are an algebra for describing the same kinds of patterns that can be described by automata
* Regular expressions can be converted to automata and vice versa



**Program written in C which examines a sequence of letters that contain aeiou which represents the Automaton above**

#include <stdio.h>

#define TRUE 1

#define FALSE 0

typedef int BOOLEAN;

BOOLEAN findChar(char \*\*pp, char c){

(while (\*\*pp != c && \*\*pp != ’\0’)

(\*pp)++;

if (\*\*pp == ’\0’)

return FALSE;

else {

(\*pp)++;

return TRUE;}

}

BOOLEAN testWord(char \*p){

/\* state 0 \*/

if (findChar(&p, ’a’))

/\* state 1 \*/

if (findChar(&p, ’e’))

/\* state 2 \*/

if (findChar(&p, ’i’))

/\* state 3 \*/

if (findChar(&p, ’o’))

/\* state 4 \*/

if (findChar(&p, ’u’))

/\* state 5 \*/

return TRUE;

return FALSE;

}

main(){

printf("%d\n", testWord("abstemious"));

}

## Thompsons Construction Algorithm