Regular Expression

Regular Expression

- A regular expression is a set of characters or pattern which is used to find substring in given string.

- Example:

✓Extracting hashtags from a given string

✓Getting Email Id or phone number from a larger unstructured text content

Basics of Regular Expression

- **Bracket([])**
- **Dash (–)**
- **Caret (^)**
- **Question Mark (?)**
- **Period (.)**

Basics of Regular Expression

1. Brackets ([]) :

- They are used to specify a disjunction of characters.
- For instance using brackets to put w/W allow us to return W or w.

- **Example**
  - ✓/[Ww]oodchuck/ -> Woodchuck or woodchuck
  - ✓/[abc]/ -> ‘a’ or ‘b’ or ‘c’
  - ✓/[1234567890]/ -> Any digit

Regex	Match	Example Patterns
/[wW]oodchuck/	Woodchuck or woodchuck	“Woodchuck”
/[abc]/	‘a’, ‘b’, *or* ‘c’	“In uomini, in soldati”
/[1234567890]/	any digit	“plenty of 7 to 5”

Figure 2.2 The use of the brackets [] to specify a disjunction of characters.

Basics of Regular Expression

2. Dash (-)

- To specify a range.
- For instance putting A-Z in brackets allows R to return all matches of an upper case letters.

- **Example**
  - ✓[0-9] -> matches single digit
  - ✓/[A-Z]/ ->matches uppercase letters
  - ✓/[a-z]/-> matches lower case letters

Regex	Match	Example Patterns Matched
/[A-Z]/	an upper case letter	"we should call it 'Drenched Blossoms' "
/[a-z]/	a lower case letter	"my beans were impatient to be hoed!"
/[0-9]/	a single digit	"Chapter 1: Down the Rabbit Hole"

Figure 2.3 The use of the brackets [] plus the dash - to specify a range.

Basics of Regular Expression

3. Caret (^)

- It can be used for negation or just to mean ^.

Example

- ✓/[^A-Z]/-> not an upper case letters
- ✓/[^a-z]/-> not an lower case letters
- ✓/[^Ss]/-> Neither S nor s
- ✓/[^\.]/-> Not a period
- ✓/[e^]/->Either e or ^
- ✓/[a^b]/-> The pattern a^b

Regex	Match (single characters)	Example Patterns Matched
/[^A-Z]/	not an upper case letter	"Oyfn pripetchik"
/[^Ss]/	neither 'S' nor 's'	"I have no exquisite reason for't"
/[^.]/	not a period	"our resident Djinn"
/[e^]/	either 'e' or '^'	"look up ^ now"
/a^b/	the pattern 'a^b'	"look up a^ b now"

Figure 2.4 The caret ^ for negation or just to mean ^. See below re: the backslash for escaping the period.

Basics of Regular Expression

4. Question marks (?)

- It marks optionality of the previous expression.
- For instance putting a? at the end of chucks returns results for woodchuck (without an s) and woodchucks (with an s)

- **Example**
  - ✓/woodchucks?/ -> woodchuck/woodchucks
  - ✓/colou?r/ -> color or colour

Basics of Regular Expression

5. Period(.)

- Used to specify any characters between expressions
- For instance putting beg.n will return begin or began

- **Example**
  - ✓/beg.n -> match any characters between a to z

Anchor

- They are used to assert something about string or the matching process.

- **Example:** ^ and $
- ^The : Matches any string that starts with ‘The’.
- End$: Matches any string that ends with ‘End’.

Regex	Match
^	start of line
$	end of line
\b	word boundary
\B	non-word boundary

Figure 2.7 Anchors in regular expressions.

Character Classes

\d	**Returns a match where the string contains digits (numbers from 0-9)**
\D	Returns a match where the string DOES NOT contain digits
\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	Returns a match where the string DOES NOT contain any word characters
\s	Returns a match where the string contains a white space character
\S	Returns a match where the string DOES NOT contain a white space character
\Z	Returns a match if the specified characters are at the end of the string
\A	Returns a match if the specified characters are at the beginning of the string

Quantifier (* + ? And {})

- abc* : ab followed by zero or more c.
- abc+ : ab followed by one or more c.
- abc? : ab followed by one or zero c.
- abc{2} : ab followed by 2c.
- abc{2,} : ab followed by 2 or more c.
- abc{2,5} : ab follower by 2 up to 5c.
- A{bc}* : a followed by zero or more copies of the sequence bc.

Program 1

```
import re
txt = "The rain in Spain"
x = re.search("^The.*Spain$", txt)
print (x)
```

Program 2

```
import re

txt = "The rain in Spain"
x = re.search("\s", txt)

print("The first white-space character is located in position:", x.start())
```

Program 3

```
import re

txt = "The rain in Spain"
x = re.search("Portugal", txt)
print(x)
```

Program 4

```
import re

txt = "The rain in Spain"
x = re.split("\s", txt)
print(x)
```

Program 5

```
import re

txt = "The rain in Spain"
x = re.findall("ai", txt)
print(x)
```

Program 6

Program 7

THANK YOU