Regular Expressions

Natural Language Processing

Daniel Ortiz Martínez, David Buchaca Prats

Regular Expressions

- Generate a regular expression using `p = re.compile(r'regexp')`
  - `p` will be a `re.Pattern` object that we can apply to any string
  - `p.findall(s)` returns a list containing all substrings within `s` that safisfy our regular expression
  - `p.finditer(s)` returns a generator. Each element of the generator is an `SRE_Match` object which contains:
    - `span(x,y)` indicates the start position `x` and end position `y` (of `s`)
    - `match='...'` indicates the string that satisfies the regular expression

Special Regex Characters

- `'.'` Matches any character except new line.
- `'d'` Matches any digit (0-9). Equivalent to `'[0-9]'`
- `'D'` Matches any NON digit. Equivalent to `'[^0-9]'`
- `'w'` Matches any "word character". Equivalent to `'[a-zA-Z0-9_]'`
- `'W'` Matches any NON alphanumeric character
  - Equivalent to `'[^a-zA-Z0-9_]'`
- `'s'` Matches any whitespace (space, tab, newline)
  - Equivalent to `'[\t\n\r\f\v]'`
- `'S'` Matches any NON whitespace (space, tab, newline)
  - Equivalent to `'[^ \t\n\r\f\v]'`

Quantifiers

- Quantifiers are operators that are applied to the preceding symbol
- `'*'` previous symbol appears 0 or more matches
- `'+'` previous symbol appears 1 or more matches
- `'?'` previous symbol appears at most one (0 or 1)
- `'{k}'` previous symbol repeated `k` exact matches
- `'{min,max}'` previous symbol appears between `min` and `max` times
    - For example `'{2,8}'` would match numbers between 2 and 8

Quantifiers

- `"r'a.*'"` the `'*'` refers to `'.'` making this expression get triggered when `'a'` is followed by any character
- `"r'd{4}'"` the `'{4}'` refers to `'d'`, making this regular expression get triggered with 4 consecutive digits
- Example:

```
aux="The girl who loved the cat ended up with a catwomen costume"
```

```
p = re.compile(r'cat\s')
p.findall(aux)
['cat ']
p = re.compile(r'cat\s*')
p.findall(aux)
['cat ', 'cat']
```

Regex Example

- Let us consider the following example:

```
aux="The girl who loved the cat ended up with a catwomen costume"
```

```
p = re.compile(r'cat')
p.findall(aux)
['cat', 'cat']
p = re.compile(r'cat\s')
p.findall(aux)
['cat ']
p = re.compile(r'cat.*')
p.findall(aux)
['cat ended up with a catwomen costume']
```

Anchors

- `'\b'` Matches any word boundary
- `'\B'` Matches any NON word boundary
- `'^'` Matches beginning of a string
- `'[^a-e]'` negates the character set `'a-e'`
- `'$'` Matches a position that is end of a string

Character Sets

- `'[]'` allows us to specify sets of symbols
- `'[ab3-]'` matches any character `'a'` or `'b'` or `'3'` or `'-'`
- `'[a-g]'` matches any character `'a'` to `'g'` such as `'b'`,`'c'`,...,`'g'`
- `'[A-G]'` matches any character `'A'` to `'G'` such as `'B'`,`'C'`,...,`'G'`
- `'[a-zA-G]'` matches any lowercase character and any uppercase character from `'A'` to `'G'`
- `'[d1-d2]'` matches any digit between `'d1'` and `'d2'`. For example `'[0-5]'` matches `'0'`,`'1'`,`'2'`,`'3'`,`'4'`,`'5'`

Examples

- `abc*` matches a string that has `ab` followed by zero or more `c`
- `abc+` matches a string that has `ab` followed by one or more `c`
- `abc?` matches a string that has `ab` followed by zero or one `c`
- `abc{2}` matches a string that has `ab` followed by 2 `c`
- `abc{2,}` matches a string that has `ab` followed by 2 or more `c`
- `abc{2,5}` matches a string that has `ab` followed by 2 up to 5 `c`
- `a(bc)*` matches a string that has `a` followed by zero or more copies of the sequence `bc`
- `a(bc){2,5}` matches a string that has `a` followed by 2 up to 5 copies of the sequence `bc`