RegularExpressionsFall24

September 10, 2024

0.0.1 Metacharacters for regular expressions

`+` means one or more repetitions of the symbol or group before it.

Let $L = \{so, soo, sooo, ...\}$ be the language of all strings that start with symbol 's' and continue with **one ore more** 'o' symbols.

A regular expression that describes (generates) this languages is `so+`.

[ ]:
```
import re

p = re.compile('so+')

m = p.match('sooo good!')
print(m)
```

```
<re.Match object; span=(0, 4), match='sooo'>
```

[ ]:
```
m.span()
```

[ ]:
```
(0, 4)
```

[ ]:
```
m.group()
```

[ ]:
```
'sooo'
```

[ ]:
```
m.start()
```

[ ]:
```
0
```

[ ]:
```
m.end()
```

[ ]:
```
4
```

[ ]:
```
m.span()[0]
```

[ ]:
```
0
```

[ ]:
```
m.span()[1]
```

[ ]:
```
4
```

[ ]:
```
simple = re.compile('so')
s = simple.match('sooo good.')
print(s)
```

```
<re.Match object; span=(0, 2), match='so'>
```

[ ]:
```
s = simple.match('this is so, sooo good.')
print(s)
```

```
None
```

[ ]:
```
s = simple.search('this is so, sooo good.')
print(s)
```

```
<re.Match object; span=(8, 10), match='so'>
```

[ ]:
```
all = simple.findall('this is so, sooo good.')
print(all)
```

```
['so', 'so']
```

[ ]:
```
p = re.compile('so+')
all = p.findall('this is so, sooo good.')
print(all)
```

```
['so', 'sooo']
```

[ ]:
```
# RE matching is greedy.
p = re.compile('so+')
m = p.search('this is so, sooo good.')
# keep searching for a match until there is no match.
while m:
  print(m.span(), m.group())
  m = p.search('this is so, sooo good.', pos = m.end())
```

```
(8, 10) so
(12, 16) sooo
```

* means zero or more repetitions of the symbol or group before it.

? means optional (zero or one occurences of the symbol or group before it)

[ ]:
```
p = re.compile('hello!?')
all = p.findall('he said "hello!" to me, and just "hello" to her.')
print(all)
```

```
['hello!', 'hello']
```

[ ]:
```
p = re.compile('hello!*')
all = p.findall('he said "hello" to him, "hello!" to me, and "hello!!!!!" to␣
 ↪her.')
print(all)
```

```
['hello', 'hello!', 'hello!!!!!']
```

[ ]:
```
p = re.compile('hello!*')
all = p.findall('"Hello", he said to him, "hello!" to me, and "hello!!!!!" to␣
↪her.')
print(all)
```

```
['hello!', 'hello!!!!!']
```

0.0.2 Brackets are RE metacharacters

[<symbols>] matches any symbol from the list of <symbols>

Let $\mathcal{L} = \{hello, hello!, hello!!, ..., Hello, Hello!, Hello!!, ...\}$

A regular expression that generates $\mathcal{L}$ is `[hH]ello!*`

[ ]:
```
p = re.compile('[Hh]ello!*')
all = p.findall('"Hello", he said to him, "hello!" to me, and "hello!!!!!" to␣
↪her.')
print(all)
```

```
['Hello', 'hello!', 'hello!!!!!']
```

`\[^\<symbols\>\]` matches any symbols that is NOT in the list `<symbols>`

For example, `\[^abcde\]` matches any symbols that is not a, b, c, d, or e.

[ ]:
```
p = re.compile('[^Hh]!ello*')
all = p.findall('"Yello", he said to him, "hello!" to me, and "shmello!!!!!" to␣
↪her.')
print(all)
```

```
['Yello', 'mello!!!!!']
```

[ ]:
```
p = re.compile('the')
all = p.findall('He caught the dog that chased the white cat.')
print(all)
```

```
['the', 'the']
```

[ ]:
```
p = re.compile('the')
all = p.findall('He caught the dog that bit them.') # => the RE generates /␣
↪covers too many strings.
print(all)
```

```
['the', 'the']
```

0.0.3 Parantheses are metacharacters too

[ ]:
```
p = re.compile(' the ')
all = p.findall('He caught the dog that bit them.') # => the RE generates /␣
 ↪covers too many strings.
print(all)
```

```
[' the ']
```

[ ]:
```
p = re.compile(' (the) ')
all = p.findall('He caught the dog that bit them.')
print(all)
```

```
['the']
```

[ ]:
```
p = re.compile(' ([Tt]he) ')
all = p.findall('The man caught the dog that bit them.')
print(all)
```

```
['the']
```

0.0.4 The pipe symbol | is a metacharacter

It is used as a disjuntion (logical or) between items in a group.

When ^ is used outside brackets, it indicates the beginning of the string.

[ ]:
```
p = re.compile('[Ww]oodchuck|[Gg]roundhog')
matches = p.findall('The woodchuck appears at the beginning in the movie␣
 ↪Groundhog Day')
matches
```

[ ]:
```
['woodchuck', 'Groundhog']
```

[ ]:
```
p = re.compile('^The')
all = p.findall('The man caught the dog that bit them.')
print(all)
```

```
['The']
```

The dot . is a metacharacter that maches anything.

[ ]:
```
# The language of all strings that start with two a's and end with 2 b's.
p = re.compile('aa.*bb')
```

[ ]:
```
p = re.compile('gr[aeiou]+vy')
all = p.findall('So groovy, greavy, grouvy, greenvy, greeeeeeouioiuoavy!')
print(all)
```

```
['groovy', 'greavy', 'grouvy', 'greeeeeeouioiuoavy']
```

[ ]:
```
# All capital letters
p = re.compile('[ABCDEFGHIJKLMNOPQRSTUVWXYZ]')
q = re.compile('[A-Z]')
```

Any digit would be `[0-9]`

Any digit between 2 and 8 would be `[2-8]`

[ ]:
```
p = re.compile('[1-5b-f]+')
all = p.findall("Let's try it with beeb, beed1-dde4, and b66ed.")
print(all)
```

```
['e', 'beeb', 'beed1', 'dde4', 'd', 'b', 'ed']
```

[ ]:
```
p = re.compile('[1-56-8]')
all = p.findall("Numbers 56 and 1 and 2 and 34")
print(all)
```

```
['5', '6', '1', '2', '3', '4']
```

[ ]:
```
p = re.compile('[gG]ro+vy')
app = p.findall("So groovy, baby, but she is definitely groooooovy!")
print(app)
```

```
['groovy', 'groooooovy']
```

[ ]:
```
def create_vocab(text):
  d = {}
  for word in text:
    if word in d:
      d[word] += 1
    else:
      d[word] = 1

  return d

text = ['one', 'two', 'four', 'joe', 'one', 'joe', 'one']
create_vocab(text)
```

[ ]:
```
{'one': 3, 'two': 1, 'four': 1, 'joe': 2}
```

[ ]:
```
def create_vocab(text):
  d = {}
  for word in text:
    d[word] = d.get(word, 0) + 1

  return d

text = ['one', 'two', 'four', 'joe', 'one', 'joe', 'one']
create_vocab(text)
```

[ ]:
```
{'one': 3, 'two': 1, 'four': 1, 'joe': 2}
```

[ ]:
```
def create_vocab(text):
  d = {}
  for word in text:
    d[word] = d[word] + 1 if d.get(word) else 1
  return d

text = ['one', 'two', 'four', 'joe', 'one', 'joe', 'one']
create_vocab(text)
```

[ ]:
```
{'one': 3, 'two': 1, 'four': 1, 'joe': 2}
```

V = {a, b, c, d, e, u, y, _}

[ ]:
```
import re

p = re.compile('[A-Za-z\-]+')
p.findall("this is a long-string with a few numbers, such as 12, 12.4 and 5")
```

[ ]:
```
['this',
 'is',
 'a',
 'long-string',
 'with',
 'a',
 'few',
 'numbers',
 'such',
 'as',
 'and']
```

0.1 Non-capturing groups

Use `(?: <pattern>)` to avoid capturing behavior.

[ ]:
```
p = re.compile('baob(?:ab)+')
matches = p.findall('I saw a baobab next to a baobabab')
matches
```

[ ]:
```
['baobab', 'baobabab']
```

0.2 More examples

[ ]:
```
p = re.compile('colour')
p.sub('color', 'I like bright colours, and I am partial to dark colours.')
```

[ ]:
```
'I like bright colors, and I am partial to dark colors.'
```

[ ]:
```
p = re.compile('wa+y')
p.sub('way', 'He is on his way, but he is still waaaay to far, and waaaaaaay to␣
↪unprepared.')
```

[ ]:
```
'He is on his way, but he is still way to far, and way to unprepared.'
```

[ ]:
```
p = re.compile('( [0-9]+ )', re.VERBOSE)
p.sub(r'<\1> extra', '10 whiseky bottles and 35 boxes of gold')
```

[ ]:
```
'<10> extra whiseky bottles and <35> extra boxes of gold'
```

0.3 Pattern substitutions and group references

[ ]:
```
p = re.compile(r'the (.*)er they (.*), the \1er we \2')
m = p.match('the faster they ran, the faster we ran')
m
```

[ ]:
```
<re.Match object; span=(0, 38), match='the faster they ran, the faster we ran'>
```

[ ]:
```
p = re.compile(r'the (.*)er they (.*), the \1er we \2')
m = p.match('the faster they ran, the faster we jumped')
print(m)
```

```
None
```

[ ]:
```

```