

# RegularExpressions

September 4, 2025

## 0.0.1 Metacharacters for regular expressions

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

Let  $L = \{so, soo, sooo, \dots\}$  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+`.

```
[37]: import re

p = re.compile('so+')

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

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

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

```
[38]: (0, 4)
```

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

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

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

```
[40]: 0
```

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

```
[41]: 4
```

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

```
[42]: 0
```

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

```
[43]: 4
```

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

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

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

None

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

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

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

['so', 'so']

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

['so', 'sooo']

```
[49]: # 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 occurrences of the symbol or group before it)

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

['hello!', 'hello']

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

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

```
[52]: 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  $L = \{hello, hello!, hello!!, \dots, Hello, Hello!, Hello!!, \dots\}$

A regular expression that generates  $L$  is `[hH]ello!*`

```
[53]: 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.

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

```
[]
```

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

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

```
[56]: 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

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

```
[' the ']
```

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

```
['the']
```

```
[59]: 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 disjunction (logical or) between items in a group.

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

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

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

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

```
['The']
```

The dot . is a metacharacter that matches anything.

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

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

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

```
[64]: # 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]

```
[65]: 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']
```

```
[66]: 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']
```

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

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

```
[70]: 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.

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

```
print(matches)

# with this re behavior:
p = re.compile('baob(?:ab)+')
matches = p.findall('I saw a baobab next to a baobabab')
print(matches)
```

```
['ab', 'ab']
['baobab', 'baobabab']
```

## 0.2 More examples

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

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

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

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

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

```
[76]: '<10> extra whiseky bottles and <35> extra boxes of gold'
```

## 0.3 Pattern substitutions and group references

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

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

```
[78]: 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

## 0.4 More examples from Sep 4, 2025

```
[79]: import re

      # Match alphanumeric strings that have no uppercase letters, that start and end
      ↪with 'z'
```

```

pos1 = "z123abcz"
neg2 = "za&abz"
neg3 = "zdsckfh"

# First try, it will have false positives (wrongly matches neg2).
p = re.compile('z.*z')
m1 = p.match(pos1)
m2 = p.match(neg2)
m3 = p.match(neg3)
print(m1, m2, m3, sep = '\n')

```

```

<re.Match object; span=(0, 8), match='z123abcz'>
<re.Match object; span=(0, 6), match='za&abz'>
None

```

```
[81]: m1.span(), m1.group(), m1.start(), m1.end()
```

```
[81]: ((0, 8), 'z123abcz', 0, 8)
```

```

[82]: # Second try, correct pattern, no false positives.
p = re.compile('z[a-zA-Z0-9]*z')
print(pos1, neg2, neg3, end = '\n')
m1 = p.match(pos1)
m2 = p.match(neg2)
m3 = p.match(neg3)
print(m1, m2, m3, sep = '\n')

```

```

z123abcz za&abz zdsckfh
<re.Match object; span=(0, 8), match='z123abcz'>
None
None

```

```

[83]: # Equivalent pattern using '\w' to match alphanumeric characters.
p = re.compile(r'z\w*z')
print(pos1, neg2, neg3, end = '\n')
m1 = p.match(pos1)
m2 = p.match(neg2)
m3 = p.match(neg3)
print(m1, m2, m3, sep = '\n')

```

```

z123abcz za&abz zdsckfh
<re.Match object; span=(0, 8), match='z123abcz'>
None
None

```

```

[84]: # Match any lowercase letter other than 'q'.
p = re.compile('[a-pr-z]')

# Match lowercase alphanumeric strings that start and end with the same symbol.

```

```
p = re.compile(r'([a-pr-z]) [a-pr-z]* \1', re.VERBOSE)
m = p.match('zbcdz')
print(m)
print(m.group())
```

```
<re.Match object; span=(0, 5), match='zbcdz'>
zbcdz
```

```
[85]: m = p.match('zbcd')
print(m)
```

```
None
```

```
[ ]:
```