regularexpressions

February 3, 2026

[2]:
```
import re
```

[ ]:
```
# search==> returns first match object or none
text="My phone number is 8309644254"
res=re.search(r'\d{10}',text)

print(res.group()) # Output: 8309644254
```

```
8309644254
```

[3]:
```
# match --> match only at the beginning of the string
text="8309644254 is my phone number"
res=re.match(r'\d{10}',text)
print(res.group()) # Output: 8309644254
print(re.match(r'is',text)) # Output: None
```

```
8309644254
None
```

[4]:
```
# full match --> match entire string
print(re.fullmatch(r'\d{10}',"8309644254")) # Output: <re.Match object;␣
 ↪span=(0, 10), match='8309644254'>
```

```
<re.Match object; span=(0, 10), match='8309644254'>
```

[7]:
```
# findall --> returns all matches in a list
text="My phone numbers are 8309644254 and 9876543210"
res=re.findall(r'\d{10}',text)
print(res) # Output: ['8309644254', '9876543210']
```

```
['8309644254', '9876543210']
```

[8]:
```
# re.sub() --> replace patter
text="My phone number is 8309644254"
print(re.sub(r'\d','*',text)) # Output: My phone number is **********
```

```
My phone number is **********
```

[ ]:
```
# split using pattern
text="apple,banana;orange|grape"
```

```
print(re.split(r'[,;|]',text)) # [.;|] used as delimiters
```

```
['apple', 'banana', 'orange', 'grape']
```

Match every object

When search/match/fullmatch works → returns Match Object

common methods of Match Object: - group() –> returns the matched string - start() –> returns starting index of matched string - end() –> returns ending index of matched string - span() –> returns tuple of (start,end) index of matched string

() –> Grouping

()–> used to capture group

(?P…) –> named capturing group

(?:…) –> non-capturing group

[11]:
```
text = "Date: 2026-01-31"
m = re.search(r"(\d{4})-(\d{2})-(\d{2})", text)

print(m.group(1))  # year
print(m.group(2))  # month
print(m.group(3))  # day
```

```
2026
01
31
```

[ ]:
```
m.group()  # full match
m.groups()  # tuple of all groups
```

[ ]:
```
('2026', '01', '31')
```

[14]:
```
print(m.groups())
# ('2026', '01', '31')
```

```
('2026', '01', '31')
```

[16]:
```
# Named Groups(?P<name>)
m=re.search(r"(?P<year>\d{4})-(?P<month>\d{2})-(?P<day>\d{2})", "Date:␣
↪2026-01-31")
print(m.group("year"))  # Output: 2026
print(m.group("month")) # Output: 01
print(m.group("day"))   # Output: 31
print(m.groupdict())   # Output: {'year': '2026', 'month': '01', 'day': '31'}
```

```
2026
01
31
{'year': '2026', 'month': '01', 'day': '31'}
```

[17]:
```
# Non -capturing Groups (?:...)
re.search(r"(?:Mr|Mrs)\.?\s[A-Z]\w+", "Mr Smith")  # Matches 'Mr Smith'
```

[17]:
```
<re.Match object; span=(0, 8), match='Mr Smith'>
```

[13]:
```
# regex
# \ means escape/ special character
# \. means literal dot
# \w --> word character [a-zA-Z0-9_]
# \d --> digit [0-9]
# \s --> whitespace [ \t\n\r\f\v]
# \W --> non-word character [^a-zA-Z0-9_]
# \* --> literal *

print(re.search(r"/","a/b"))


# ^ start of the string (starts witj character or word )
print(re.search(r"^Hello","Hello World"))  # Matches
print(re.search(r"^World","Hello World"))  # None

# $-- end of the string
print(re.search(r"World$","Hello World"))  # Matches
print(re.search(r"Hello$","Hello World"))  # None

# . --> any character except newline
print(re.search(r"h.t","hat"))  # Matches

# * --> 0 or more ( a*c --> a followed by 0 or more a's followed by c)
print(re.search(r"a*c","aaac"))

# + --> 1 or more( a+c --> a followed by 1 or more a's followed by c)
# + means must appear at least once
print(re.search(r"a+c","aaac"))

# ? --> 0 or 1 ( a?c --> a followed by 0 or 1 a's followed by c)
print(re.search(r"a?c","ac"))
print(re.search(r"a?c","c"))
```

```
<re.Match object; span=(1, 2), match='/'>
<re.Match object; span=(0, 5), match='Hello'>
None
<re.Match object; span=(6, 11), match='World'>
None
<re.Match object; span=(0, 3), match='hat'>
<re.Match object; span=(0, 4), match='aaac'>
<re.Match object; span=(0, 4), match='aaac'>
<re.Match object; span=(0, 2), match='ac'>
```

```
<re.Match object; span=(0, 1), match='c'>
```

[ ]:
```
# { } --> exact count/range(a{m,n}c --> a followed by m to n a's followed by c)

# \d{10} --> exactly 10 digits
# \d{2,4} --> between 2 to 4 digits
# \d{3,} --> 3 or more digits
print(re.search(r"a{2,4}c","aaac"))  # Matches
print(re.search(r"a{4,}c","aaac"))   # None
```

```
<re.Match object; span=(0, 4), match='aaac'>
None
```

[18]:
```
# | --> OR operator
print(re.search(r"cat|dog","I have a dog"))  # Matches 'dog'
print(re.search(r"cat|dog","I have a cat"))  # Matches 'cat
```

```
<re.Match object; span=(9, 12), match='dog'>
<re.Match object; span=(9, 12), match='cat'>
```

[20]:
```
# [ ] --> character set
# match one of many
# [aeiou] --> matches any vowel
print(re.search(r"[aeiou]","sky"))  # None
print(re.search(r"[aeiou]","apple"))  # Matches 'a'
print(re.search(r"[0-9]","Version 2.0"))  # Matches '2'

# inside [], symbol lose power
# [^0-9] --> matches any non-digit character
print(re.search(r"[^0-9]","Version 2.0"))  #
```

```
None
<re.Match object; span=(0, 1), match='a'>
<re.Match object; span=(8, 9), match='2'>
<re.Match object; span=(0, 1), match='V'>
```

[21]:
```
# () --> Group
# used to extract data
# apply quantifier to multiple chars

# (ab)+ ==> matches one or more occurrences of "ab"
print(re.search(r"(ab)+","abababxyz"))  # Matches 'ababab'

# (\d{2})-(\d{2}) ==> matches patterns like "12-34"
m = re.search(r"(\d{2})-(\d{2})", "Date: 12-34")
print(m.group(1))  # Output: 12
print(m.group(2))  # Output: 34
```

```
<re.Match object; span=(0, 6), match='ababab'>
12
```

34

[22]:
```
# \d+ - digit (same as [0-9])
# \D+ -->not digit
# \w --> word character (same as [a-zA-Z0-9_])

# \W --> non-word character (same as [^a-zA-Z0-9_])
print(re.search(r"\D+","Version 2.0"))  # Matches 'Version '
```

```
<re.Match object; span=(0, 8), match='Version '>
```

[ ]:
```
# (?=.) ==> Positive lookahead assertion, ensures that what follows matches the␣
 ↪pattern inside the parentheses without including it in the match

# examples
# (?=.*[A-Z]) --> at least one uppercase
# (?=.*[a-z]) --> at least one lowercase
# (?=.*\d)    --> at least one digit
# (?=.*[@#$%^&+=]) --> at least one special character
```

[ ]:
```
# \s --> whitespace character (space, tab, newline)
# \S --> non-whitespace character

# \ ex: \s+, \S+
```

[23]:
```
# Examples
# Requirement:
# username# letters, digits, dot =[A-Za-z0-9._]+ --> here _ means actual␣
 ↪underscore
# @ --> @
# domain --> [a-zA-Z]+
# .com or .in --> \.(com|in)+ ==> \. means actual dot

pattern=r"^[a-zA-Z0-9._]+@[a-zA-Z]+\.(com|in)+$"
print(re.search(pattern,"example.user_123@domain.com"))  # Matches
```

```
<re.Match object; span=(0, 27), match='example.user_123@domain.com'>
```

[25]:
```
# Indian mobile number
# general mobile number: \d{10}
# starts with 6-9,total 10 digits
pattern=r"^[6-9]\d{9}$"
print(re.search(pattern,"8309644254"))  # Matches
```

```
<re.Match object; span=(0, 10), match='8309644254'>
```

[26]:
```
# password(strong )
# Min 8 chars
# 1 uppercase
```

```
# 1 lowercase
# 1 digit
# 1 special

pattern = r"^(?=.*[A-Z])(?=.*[a-z])(?=.*\d)(?=.
  ↪*[@#$%^&+=])[A-Za-z\d@#$%^&+=]{8,}$"  # at least 8 chars from the given set
```