Python Regular Expressions (RegEx) - Complete Guide

Table of Contents

---

1. Introduction to Regular Expressions {#introduction}

What is RegEx? A Regular Expression (RegEx) is a sequence of characters that forms a search pattern. It's a powerful tool for:

- Pattern matching in strings
- Text validation (emails, phone numbers, etc.)
- Data extraction and cleaning
- String manipulation and replacement

Why use RegEx?

- Efficient text processing
- Complex pattern matching
- Data validation
- Text parsing and extraction

---

2. RegEx Module in Python {#module}

Python provides the built-in `re` module for working with regular expressions.

python

```python
import re

# Basic example
text = "The rain in Spain"
pattern = "^The.*Spain$"
match = re.search(pattern, text)
print(match)  # <re.Match object; span=(0, 17), match='The rain in Spain'>
```

---

3. RegEx Functions {#functions}

The `re` module provides several key functions:

3.1 `re.search(pattern, string, flags=0)`

Searches for the first occurrence of a pattern in a string.

python

```python
import re

text = "The rain in Spain"
result = re.search("rain", text)
if result:
    print(f"Found '{result.group()}' at position {result.start()}-{result.end()}")
# Output: Found 'rain' at position 4-8
```

3.2 `re.match(pattern, string, flags=0)`

Checks if the pattern matches at the beginning of the string.

python

```python
import re

text = "Hello World"
result = re.match("Hello", text)
print(result.group() if result else "No match")  # Output: Hello

result = re.match("World", text)
print(result.group() if result else "No match")  # Output: No match
```

3.3 `re.findall(pattern, string, flags=0)`

Returns all non-overlapping matches as a list.

python

```python
import re

text = "The rain in Spain falls mainly on the plain"
result = re.findall("ain", text)
print(result)  # Output: ['ain', 'ain', 'ain', 'ain']

# Finding all words starting with 'S' or 's'
result = re.findall(r'\b[Ss]\w+', text)
print(result)  # Output: ['Spain']
```

3.4 `re.split(pattern, string, maxsplit=0, flags=0)`

Splits the string by the occurrences of the pattern.

python

```python
import re

text = "apple,banana;orange:grape"
result = re.split('[,;:]', text)
print(result)  # Output: ['apple', 'banana', 'orange', 'grape']

# Limit splits
result = re.split('[,;:]', text, maxsplit=2)
print(result)  # Output: ['apple', 'banana', 'orange:grape']
```

3.5 `re.sub(pattern, replacement, string, count=0, flags=0)`

Replaces occurrences of the pattern with a replacement string.

python

```python
import re

text = "The rain in Spain"
result = re.sub("Spain", "France", text)
print(result)  # Output: The rain in France

# Replace with count limit
result = re.sub("a", "X", text, count=2)
print(result)  # Output: The rXin in SpXin
```

3.6 `re.compile(pattern, flags=0)`

Compiles a regular expression pattern for reuse.

python

```python
import re

# Compile for efficiency when using multiple times
pattern = re.compile(r'\b\w+@\w+\.\w+\b')
emails = ["test@example.com", "invalid-email", "user@domain.org"]

for email in emails:
    if pattern.search(email):
        print(f"{email} is valid")
```

---

4. Metacharacters {#metacharacters}

Metacharacters have special meanings in regular expressions:

Character	Description	Example	Matches
`.`	Any character (except newline)	`a.c`	"abc", "axc", "a1c"
`^`	Start of string	`^Hello`	"Hello world"
`$`	End of string	`world$`	"Hello world"
`*`	Zero or more occurrences	`ab*c`	"ac", "abc", "abbc"
`+`	One or more occurrences	`ab+c`	"abc", "abbc" (not "ac")
`?`	Zero or one occurrence	`ab?c`	"ac", "abc" (not "abbc")
`{n}`	Exactly n occurrences	`a{3}`	"aaa"
`{n,m}`	Between n and m occurrences	`a{2,4}`	"aa", "aaa", "aaaa"
`[]`	Set of characters	`[abc]`	"a", "b", or "c"
`` ` ``	`` ` ``	Either or	`cat
`()`	Group	`(abc)+`	"abc", "abcabc"
`\`	Escape character	`\.`	Literal dot "."

Practical Examples:

python

```python
import re

# Any character (.)
text = "cat, bat, rat"
result = re.findall(r'.at', text)
print(result)  # Output: ['cat', 'bat', 'rat']

# Start and end anchors (^, $)
texts = ["hello world", "world hello", "hello"]
pattern = r'^hello'
for text in texts:
    if re.search(pattern, text):
        print(f"'{text}' starts with 'hello'")

# Quantifiers (*, +, ?)
text = "a aa aaa aaaa"
print(re.findall(r'a*', text))    # Zero or more
print(re.findall(r'a+', text))    # One or more
print(re.findall(r'a?', text))    # Zero or one

# Grouping ()
text = "123-456-7890"
pattern = r'(\d{3})-(\d{3})-(\d{4})'
match = re.search(pattern, text)
if match:
    print(f"Area code: {match.group(1)}")
    print(f"Exchange: {match.group(2)}")
    print(f"Number: {match.group(3)}")
```

---

5. Special Sequences {#special-sequences}

Special sequences start with backslash \ and have special meanings:

Sequence	Description	Example
`\A`	Start of string	`\AThe`
`\Z`	End of string	`end\Z`
`\b`	Word boundary	`\bword\b`
`\B`	Not word boundary	`\Bword\B`
`\d`	Digit [0-9]	`\d+`
`\D`	Not digit	`\D+`
`\s`	Whitespace	`\s+`
`\S`	Not whitespace	`\S+`
`\w`	Word character [a-zA-Z0-9_]	`\w+`
`\W`	Not word character	`\W+`

Practical Examples:

python

```python
import re

# Word boundaries
text = "The word 'sword' contains 'word'"
print(re.findall(r'\bword\b', text))  # Output: ['word']
print(re.findall(r'word', text))      # Output: ['word', 'word']

# Digits and non-digits
text = "Phone: 123-456-7890"
print(re.findall(r'\d', text))        # All digits
print(re.findall(r'\d+', text))       # Digit sequences
print(re.findall(r'\D+', text))       # Non-digit sequences

# Whitespace
text = "Hello\tWorld\nPython"
print(re.split(r'\s+', text))         # Split on any whitespace

# Word characters
text = "hello_world123 @#$"
print(re.findall(r'\w+', text))       # Output: ['hello_world123']
print(re.findall(r'\W+', text))       # Output: [' @#$']
```

---

6. Sets {#sets}

Sets are defined using square brackets `[]` and match any character within the brackets:

Set	Description	Example
`[abc]`	Match a, b, or c	`[aeiou]` matches vowels
`[a-z]`	Match any lowercase letter	`[a-z]+`
`[A-Z]`	Match any uppercase letter	`[A-Z]+`
`[0-9]`	Match any digit	`[0-9]+`
`[^abc]`	Match anything except a, b, or c	`[^0-9]`
`[a-zA-Z0-9]`	Match alphanumeric characters	`[a-zA-Z0-9_]+`

Practical Examples:

python

```python
import re

# Character sets
text = "Hello World 123"
print(re.findall(r'[aeiou]', text))          # Vowels: ['e', 'o', 'o']
print(re.findall(r'[A-Z]', text))            # Uppercase: ['H', 'W']
print(re.findall(r'[0-9]', text))            # Digits: ['1', '2', '3']
print(re.findall(r'[^a-zA-Z\s]', text))      # Non-letters: ['1', '2', '3']

# Range combinations
text = "abc123XYZ"
print(re.findall(r'[a-zA-Z]', text))         # Letters only
print(re.findall(r'[a-z0-9]', text))         # Lowercase + digits

# Custom sets
text = "Price: $19.99"
print(re.findall(r'[$.\d]+', text))          # Output: ['$19.99']
```

---

7. Flags {#flags}

Flags modify how the regular expression is interpreted:

Flag	Shorthand	Description
`re.IGNORECASE`	`re.I`	Case-insensitive matching
`re.MULTILINE`	`re.M`	`^` and `$` match line boundaries
`re.DOTALL`	`re.S`	`.` matches newlines too
`re.VERBOSE`	re.X`	Allows comments and whitespace
`re.ASCII`	`re.A`	ASCII-only matching
`re.UNICODE`	`re.U`	Unicode matching (default in Python 3)

Practical Examples:

```python
import re

# Case insensitive
text = "Hello WORLD"
print(re.findall(r'hello', text, re.IGNORECASE))  # Output: ['Hello']

# Multiline
text = """First line
Second line
Third line"""
print(re.findall(r'^S\w+', text, re.MULTILINE))   # Output: ['Second']

# Dot all
text = "Hello\nWorld"
print(re.findall(r'Hello.World', text, re.DOTALL))  # Output: ['Hello\nWorld']

# Verbose (allows comments)
pattern = re.compile(r'''
    \d{3}     # Area code
    -         # Separator
    \d{3}     # Exchange
    -         # Separator
    \d{4}     # Number
''', re.VERBOSE)

phone = "123-456-7890"
print(pattern.search(phone).group())  # Output: 123-456-7890
```

---

8. Match Object {#match-object}

When a match is found, a Match object is returned with useful methods and properties:

Methods:

- `.group()` - Returns the matched string
- `.start()` - Returns start position of match
- `.end()` - Returns end position of match
- `.span()` - Returns tuple of (start, end)
- `.groups()` - Returns all groups as tuple

Properties:

- `.string` - The original string
- `.pos` - Start position of search
- `.endpos` - End position of search

Practical Examples:

python

```python
import re

# Basic match object usage
text = "The rain in Spain"
match = re.search(r'r(\w+)n', text)

if match:
    print(f"Full match: {match.group()}")      # Output: rain
    print(f"Group 1: {match.group(1)}")       # Output: ai
    print(f"Start: {match.start()}")          # Output: 4
    print(f"End: {match.end()}")              # Output: 8
    print(f"Span: {match.span()}")            # Output: (4, 8)

# Multiple groups
text = "John Doe, age 30"
pattern = r'(\w+) (\w+), age (\d+)'
match = re.search(pattern, text)

if match:
    print(f"First name: {match.group(1)}")    # John
    print(f"Last name: {match.group(2)}")     # Doe
    print(f"Age: {match.group(3)}")           # 30
    print(f"All groups: {match.groups()}")    # ('John', 'Doe', '30')
```

---

9. Practical Examples {#practical-examples}

Email Validation

python

```python
import re

def validate_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return bool(re.match(pattern, email))

emails = ["test@example.com", "invalid.email", "user@domain.co.uk"]
for email in emails:
    print(f"{email}: {validate_email(email)}")
```

Phone Number Extraction

python

```python
import re

text = "Call me at 123-456-7890 or (555) 123-4567"
patterns = [
    r'\d{3}-\d{3}-\d{4}',           # 123-456-7890
    r'\(\d{3}\) \d{3}-\d{4}'        # (555) 123-4567
]

for pattern in patterns:
    matches = re.findall(pattern, text)
    print(f"Found: {matches}")
```

URL Extraction

python

```python
import re

text = "Visit https://www.python.org or http://github.com for more info"
pattern = r'https?://[^\s]+'
urls = re.findall(pattern, text)
print(f"URLs found: {urls}")
```

Password Validation

python

```python
import re

def validate_password(password):
    """
    Password must contain:
    - At least 8 characters
    - At least one uppercase letter
    - At least one lowercase letter
    - At least one digit
    - At least one special character
    """
    if len(password) < 8:
        return False

    patterns = [
        r'[A-Z]',           # Uppercase
        r'[a-z]',           # Lowercase
        r'\d',              # Digit
        r'[!@#$%^&*(),.?":{}|<>]'  # Special char
    ]

    return all(re.search(pattern, password) for pattern in patterns)

passwords = ["Password123!", "password", "PASSWORD123", "Pass123"]
for pwd in passwords:
    print(f"{pwd}: {validate_password(pwd)}")
```

Data Extraction from Text

python

```python
import re

# Extract dates in various formats
text = "Important dates: 2023-12-25, 12/25/2023, Dec 25, 2023"

date_patterns = [
    r'\d{4}-\d{2}-\d{2}',                    # YYYY-MM-DD
    r'\d{1,2}/\d{1,2}/\d{4}',                # MM/DD/YYYY
    r'[A-Za-z]{3} \d{1,2}, \d{4}'           # Dec 25, 2023
]

for pattern in date_patterns:
    dates = re.findall(pattern, text)
    print(f"Dates found with pattern {pattern}: {dates}")
```

---

10. Common Use Cases {#use-cases}

1. Data Cleaning

python

```python
import re

# Remove extra whitespaces
text = "  Hello    World   "
cleaned = re.sub(r'\s+', ' ', text.strip())
print(f"'{cleaned}'")  # 'Hello World'

# Remove non-alphanumeric characters
text = "Hello, World! 123"
cleaned = re.sub(r'[^a-zA-Z0-9\s]', '', text)
print(cleaned)  # Hello World 123
```

2. Text Processing

python

```python
import re

# Extract hashtags from social media text
text = "Loving this #python #programming #coding session! #AI"
hashtags = re.findall(r'#\w+', text)
print(hashtags)  # ['#python', '#programming', '#coding', '#AI']

# Extract mentions
text = "Thanks @john_doe and @jane_smith for the help!"
mentions = re.findall(r'@\w+', text)
print(mentions)  # ['@john_doe', '@jane_smith']
```

3. Log File Analysis

python

```python
import re

# Parse log entries
log_entry = "2023-12-25 10:30:45 ERROR Failed to connect to database"
pattern = r'(\d{4}-\d{2}-\d{2}) (\d{2}:\d{2}:\d{2}) (\w+) (.+)'
match = re.match(pattern, log_entry)

if match:
    date, time, level, message = match.groups()
    print(f"Date: {date}, Time: {time}, Level: {level}, Message: {message}")
```

4. Configuration File Parsing

python

```python
import re

# Parse key-value pairs from config
config_text = """
database_host = localhost
database_port = 5432
debug_mode = true
"""

pattern = r'(\w+)\s*=\s*(.+)'
matches = re.findall(pattern, config_text)
config_dict = {key.strip(): value.strip() for key, value in matches}
print(config_dict)
```

Best Practices

1. **Use Raw Strings**: Always use `r''` for regex patterns to avoid escaping issues
2. **Compile Patterns**: Use `re.compile()` for patterns used multiple times
3. **Be Specific**: Make patterns as specific as possible to avoid false matches
4. **Test Thoroughly**: Test regex patterns with various inputs
5. **Use Online Tools**: Utilize regex testing tools for complex patterns
6. **Comment Complex Patterns**: Use `re.VERBOSE` flag for complex patterns
7. **Handle Edge Cases**: Always check for `None` when using `search()` or `match()`

Summary

Regular expressions are powerful tools for text processing in Python. The `re` module provides comprehensive functionality for pattern matching, text extraction, and string manipulation. Key points to remember:

- Import the `re` module to use regular expressions
- Use appropriate functions (`search`, `match`, `findall`, `split`, `sub`)
- Understand metacharacters and special sequences
- Utilize character sets and flags for flexible matching
- Work with Match objects to extract detailed information
- Practice with real-world examples to master regex patterns

Regular expressions might seem complex at first, but with practice, they become an invaluable tool for any Python programmer dealing with text data.

This guide covers the essential concepts of Python regular expressions. For more advanced topics and edge cases, refer to the official Python documentation.