Regular Expression

Premanand S

Assistant Professor
School of Electronics Engineering
Vellore Institute of Technology

premanand.s@vit.ac.in

October 8, 2024

Introduction to Regular Expressions

- A regular expression (regex) is a sequence of characters that forms a search pattern.
- It is used for pattern matching within strings.
- Common operations:
  - Searching for text
  - Replacing text
  - Validating formats
- Used across many programming languages.
- Python provides the `re` module to work with regular expressions.

Use Cases of Regular Expressions

Regular expressions are useful in:

- Validating emails
- Validating phone numbers
- Searching for patterns in text
- Web scraping
- Password validation

Metacharacters

Metacharacters are special characters in regular expressions that help in defining patterns. They allow matching, searching, and manipulating text based on specific rules.
Some commonly used metacharacters:

- . (Dot)
- ^ (Caret)
- $ (Dollar)
- * (Asterisk)
- + (Plus)
- ? (Question Mark)
- [] (Square Brackets)
- () (Parentheses)
- | (Pipe)
- {} (Braces)
- \ (Backslash)

Metacharacter: . (Dot)

- Matches any single character except the newline character `\n`.
- Example: `a.b` matches "a_b".

Example (Python Code)

```
import re
pattern = r"a.b"
match = re.match(pattern, "a_b")
print(match)
```

Metacharacter: ^ (Caret)

- Anchors the match to the start of the string.
- Example: `^abc` matches "abc" only at the start of the string.

Example (Python Code)

```
import re
pattern = r"^abc"
match = re.match(pattern, "abc123")
print(match)
```

Metacharacter: $ (Dollar)

- Anchors the match to the end of the string.
- Example: abc$ matches "abc" only at the end of the string.

Example (Python Code)

```
import re
pattern = r"abc$"
match = re.search(pattern, "123abc")
print(match)
```

Metacharacter: * (Asterisk)

- Matches zero or more of the preceding character.
- Example: `a*` matches "aaaa", "", etc.

Example (Python Code)

```
import re
pattern = r"a*"
match = re.match(pattern, "aaaa123")
print(match)
```

Metacharacter: + (Plus)

- Matches one or more of the preceding character.
- Example: a+ matches "a", "aaa", etc.

Example (Python Code)

```
import re
pattern = r"a+"
match = re.match(pattern, "aaa123")
print(match)
```

Metacharacter: ? (Question Mark)

- Matches zero or one of the preceding character (optional).
- Example: `colou?r` matches "color" or "colour".

Example (Python Code)

```
import re
pattern = r"colou?r"
match = re.match(pattern, "color")
print(match)
```

Metacharacter: [] (Square Brackets)

- Matches any single character inside the brackets.
- Example: [abc] matches "a", "b", or "c".
- Ranges can be specified, e.g., [A-Z], [0-9].

Example (Python Code)

```
import re
pattern = r"[abc]"
match = re.match(pattern, "a123")
print(match)  # Output: <re.Match object; span=(0, 1), match=
```

Metacharacter: {} (Braces)

- Specifies the number of repetitions for the preceding character.
- Example: `a{3}` matches "aaa".
- Variants:
  - `a{n}`: Exactly `n` occurrences.
  - `a{n,}`: At least `n` occurrences.
  - `a{n,m}`: Between `n` and `m` occurrences.

Example (Python Code)

```
import re
pattern = r"a{3}"
match = re.match(pattern, "aaa123")
print(match)
```

Metacharacter: () (Parentheses)

- Used for grouping parts of a regex and capturing.
- Example: `(abc)+` matches "abcabc".

Example (Python Code)

```
import re
pattern = r"(abc)+"
match = re.match(pattern, "abcabc123")
print(match)
```

Metacharacter: | (Pipe)

- Acts as an OR operator.
- Example: cat|dog matches "cat" or "dog".

Example (Python Code)

```
import re
pattern = r"cat|dog"
match = re.match(pattern, "dog123")
print(match)
```

Metacharacter: \ (Backslash)

- Escapes a metacharacter to treat it as a literal character.
- Example: \$100 matches the literal string "$100".

Example (Python Code)

```
import re
pattern = r"\$100"
match = re.match(pattern, "$100")
print(match)
```

The re Module

- Python provides the `re` module to work with regular expressions.
- This module includes several functions to search, match, replace, and manipulate strings based on patterns.
- It is highly flexible and widely used for pattern matching and text processing.

Basic Functions of the re Module

The `re` module provides several key functions to work with regular expressions:

- `re.match()`: Checks for a match only at the beginning of the string.
- `re.search()`: Searches the entire string for a match.
- `re.findall()`: Returns all matches of a pattern in the string as a list.
- `re.sub()`: Replaces all occurrences of a pattern in a string with a replacement.
- `re.split()`: Splits the string by occurrences of a pattern.

Example: re.match()

Example (Python)

re.match() tries to match the pattern at the beginning of the string.

```
import re
pattern = r'^[A-Za-z]+'
text = "Python is fun!"
match = re.match(pattern, text)
if match:
    print(f"Match found: {match.group()}")
else:
    print("No match")
```

Example: re.search()

Example (Python)

`re.search()` searches for the pattern anywhere in the string.

```
import re
pattern = r'fun'
text = "Python is fun!"
match = re.search(pattern, text)
if match:
    print(f"Match found: {match.group()}")
else:
    print("No match")
```

Example: re.findall()

Example (Python)

`re.findall()` returns all matches of the pattern as a list.

```
import re
pattern = r'\d+'
text = "There are 2 cats, 3 dogs, and 5 birds."
matches = re.findall(pattern, text)
print(f"Matches found: {matches}")
```

Example: re.sub()

Example (Python)

re.sub() replaces occurrences of the pattern with the replacement string.

```
import re
pattern = r'\s+'
text = "This is a test string."
new_text = re.sub(pattern, '-', text)
print(f"New Text: {new_text}")
```

Example: re.split()

Example (Python)

re.split() splits the string by the occurrences of the pattern.

```
import re
pattern = r'\s+'
text = "Split this string by spaces."
result = re.split(pattern, text)
print(f"Result: {result}")
```

Metacharacter: Special Sequences

- \d: Matches any digit.
- \D: Matches any non-digit.
- \w: Matches any word character (letters, digits, underscore).
- \W: Matches any non-word character.
- \s: Matches any whitespace.
- \S: Matches any non-whitespace.

Example (Python Code)

```
import re
pattern = r"\d+"
match = re.match(pattern, "123abc")
print(match)
```

Metacharacter: \d (Digit)

- Matches any digit (0-9).

Example (Python Code)

```
import re
text = "There are 3 cats and 4 dogs."
pattern = r"\d"
matches = re.findall(pattern, text)
print(matches)  # Output: ['3', '4']
```

Metacharacter: \D (Non-Digit)

- Matches any character that is not a digit.

Example (Python Code)

```
import re
text = "There are 3 cats and 4 dogs."
pattern = r"\D"
matches = re.findall(pattern, text)
print(matches)
# Output: ['T', 'h', 'e', 'r', 'e', ' ', 'a', 'r', 'e', ' ',
```

Metacharacter: `\w` (Word Character)

- Matches any alphanumeric character or underscore.

Example (Python Code)

```
import re
text = "User_name123 has logged in."
pattern = r"\w+"
matches = re.findall(pattern, text)
print(matches)
```

Metacharacter: \W (Non-Word Character)

- Matches any character that is not a word character.

Example (Python Code)

```
import re
text = "Welcome to @Python3!"
pattern = r"\W"
matches = re.findall(pattern, text)
print(matches)
```

Metacharacter: \s (Whitespace)

- Matches any whitespace character (space, tab, newline).

Example (Python Code)

```
import re
text = "Whitespace   in  between  words."
pattern = r"\s+"
matches = re.findall(pattern, text)
print(matches)
```

Metacharacter: \S (Non-Whitespace)

- Matches any character that is not a whitespace character.

Example (Python Code)

```
import re
text = "Whitespace   in  between  words."
pattern = r"\S+"
matches = re.findall(pattern, text)
print(matches)
```

Conclusion

- Regular expressions are a powerful tool for pattern matching and data manipulation.
- They are widely used in tasks like validation, text searching, web scraping, and more.
- Mastering regex helps you handle complex text-processing tasks efficiently.