Regular Expressions (Regex)

Regular expressions, or regex, are a sequence of characters that define a search pattern. They are commonly used for text matching, manipulation, and validation. Regex provides a powerful and flexible way to work with strings, allowing you to search, match, split, and replace text based on specific patterns.

Applications of Regex

Regular expressions (regex) have various applications across various fields, particularly in programming, data science, and text processing. Here are some key applications of regex:

1. Data Validation

Regex is commonly used to validate user input, ensuring that it conforms to expected formats. For example, regex can check if an email address, phone number, or credit card number is formatted correctly. This is crucial in web forms and applications to prevent invalid data entry.

2. Data Extraction

In data science and web scraping, regex is invaluable for extracting specific information from unstructured text. It can be used to pull out email addresses, URLs, or any other patterns from large datasets. For instance, regex can efficiently extract links from HTML content or find specific data points in log files.

3. Text Manipulation

Regex allows for powerful text manipulation capabilities, such as replacing substrings, splitting strings, and reformatting text. This is useful in scenarios where you need to clean or transform data, such as changing date formats or removing unwanted characters from strings.

4. Search and Replace

In text editors and programming environments, regex is often used for search-and-replace functionality. This allows users to find complex patterns in the text and replace them with new content, which can save significant time compared to manual editing.

5. Programming Language Features

Many programming languages, including Python, JavaScript, and Perl, include built-in support for regex, allowing developers to implement complex text processing features directly in their applications. This enhances the capabilities of software by enabling sophisticated string manipulation and pattern matching.

Basic Concepts of Regular Expressions

1. Literal Characters: These are simply characters that match themselves. For example, the regex cat will match the string "cat".
2. Special Characters:
    - .: Matches any single character except a new line.
    - ^: Matches the start of a string.
    - $: Matches the end of a string.
    - *: Matches 0 or more repetitions of the preceding element.
    - +: Matches 1 or more repetitions of the preceding element.
    - ?: Matches 0 or 1 repetition of the preceding element.
    - \: Escapes a special character.
3. Character Classes:
    - [abc]: Matches any one of the characters a, b, or c.
    - [a-z]: Matches any lowercase letter.
    - [A-Z]: Matches any uppercase letter.
    - [0-9]: Matches any digit.
    - \d: Matches any digit, equivalent to [0-9].
    - \D: Matches any non-digit character.

- \w: Matches any word character (alphanumeric + underscore).
- \W: Matches any non-word character.
- \s: Matches any whitespace character.
- \S: Matches any non-whitespace character.

4. Quantifiers:
   - {n}: Matches exactly n repetitions of the preceding element.
   - {n,}: Matches n or more repetitions of the preceding element.
   - {n,m}: Matches between n and m repetitions of the preceding element.
5. Groups and Capturing:
   - (abc): Matches the exact sequence "abc".
   - (a|b): Matches either "a" or "b".
   - (abc)+: Matches one or more repetitions of "abc".

Common Functions for Data Extraction

Python provides the re module for working with regular expressions. Here are some common functions from the re module:

1. re.findall(pattern, string): Returns a list of all matches in the string
2. re.search(pattern, string): Returns a match object if the pattern is found, otherwise None
3. re.split(pattern, string): Splits the string at each match of the pattern
4. re.sub(pattern, replacement, string): Replaces all matches of the pattern with the replacement string
5. re.compile(pattern): Compiles the regex pattern for later use

Example 1: Matching a Simple String

```
import re

pattern = "Python"
text = "I write code in Python."
match = re.search(pattern, text)
print("Match found:", match.group())
```

Example 2: Finding all matching occurrences

```
import re

pattern = "a"
text = "I love to code in Python and it's amazing!"
matches = re.findall(pattern, text)
print("Matches found:", len(matches))
```

Example 3: Splitting a string with white spaces

```
import re

pattern = "\s"
text = "I like Python and it is amazing!"
split_text = re.split(pattern, text)
print("Split text:", split_text)
```

Example 4: Replacing a string

```
import re

pattern = "Python"
replacement = "Java"
text = "Python is fun"
substituted_text = re.sub(pattern, replacement, text)
print("Substituted text:", substituted_text)
```

Example 5: Using the caret symbol(^) to find whether it matches the start of the string

```
import re

pattern = "^I"
```

```
text = "I love Python!"
match = re.search(pattern, text)
print("Match found:", bool(match))
```

Example 6: Using a dollar sign to check whether it matches the end of a string.

```
import re

pattern = "Great!$"
text = "Python programming is great!"
match = re.search(pattern, text)
print("Match found:", bool(match))
```

Example 7: Create a character set using [] to find matches for the string

```
import re

pattern = "[Pp]ython"
text = "I write code in python and Python!"
matches = re.findall(pattern, text)
print("Matches found:", len(matches))
```

Example 8: Using a . to match the number of characters

```
import re

pattern = "Py...n"
text = "I love Python, Pyt3on, Py45n, and Py@#n!"
matches = re.findall(pattern, text)
print("Matches found:", len(matches))
```

Example 9: Using * to find zero or more occurences of a character

```
import re
```

```
pattern = "Py.*n"
count = 0
text = ["Python coding", "Pyt3on","Java", "Py45n", "Py@#n","Pyn"]
for i in text:
  if(re.findall(pattern, i)):
     count+=1
print("Matches found:", count)
```

Example 10: using + to find one or more occurrences of a character

```
import re

pattern = "Py.+n"
count = 0
text = ["Python coding", "Pyt3on","Java", "Py45n", "Py@#n","Pyn"]
for i in text:
  if(re.findall(pattern, i)):
     count+=1
print("Matches found:", count)
```

Example 11: Using ? to find zero or one occurence of a character

```
import re

pattern = "Py.?n"
count = 0
text = ["Python coding", "Pyt3on","Java", "Py45n", "Py@#n","Pyn"]
for i in text:
  if(re.findall(pattern, i)):
     count+=1
print("Matches found:", count)
```

Example 12: Using {} to specify the number of occurrences of a character

```
import re

pattern = "Py{1,2}n"
text = "I love Pyn, Pyyn, and Pyyyn!"
matches = re.findall(pattern, text)
print("Matches found:", len(matches))
```

Example 13: Using \d to match digits

```
import re

pattern = "\d+"
text = "My phone number is 123-456-7890."
matches = re.findall(pattern, text)
print(matches)
print("Matches found:", len(matches))
```

Example 14: Using \w to match alphanumeric characters

```
import re

pattern = "\w+"
text = "I_love_Python!"
matches = re.findall(pattern, text)
print(matches)
print("Matches found:", len(matches))
```

Example 15: Using \s to match whitespace characters

```
import re

pattern = "\s+"
text = "I love Python!"
matches = re.findall(pattern, text)
```

```
print("Matches found:", len(matches))
```

Example 16: Using | as an OR operator

```
import re

pattern = "Python|Java"
text = "I write code in Python and Java!"
matches = re.findall(pattern, text)
print(matches)
print("Matches found:", len(matches))
```

Example 17: Using () to group patterns / To get phone numbers

```
import re

pattern = "(\d{3})-(\d{3})-(\d{4})"
text = "My phone number is 123-456-7890."
match = re.search(pattern, text)
if match:
  print("Area code:", match.group(1))
  print("Local exchange:", match.group(2))
  print("Line number:", match.group(3))
```

Example 18: To match emails

```
import re

text = "Contact us at example@gmail.com or home@gmail.com"
pattern = '[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}'
emails = re.findall(pattern, text)
print(emails)
```

Example 19: Extracting dates

```
import re

text = "The event will be held on 25/12/2024."
pattern = "\d{2}/\d{2}/\d{4}"

dates = re.findall(pattern, text)
print("Dates found:", dates)
```

Exercises

1. Write a regular expression to extract URLs from a string
2. Write a regular expression to extract HTML tags from a string
3. Write a regular expression to extract Kenyan Phone Numbers