Module 9: Advanced Regular Expression Techniques

Regular expressions (regex) are powerful tools for pattern matching and text manipulation in Python. Advanced regex techniques extend the capabilities of basic pattern matching to handle more complex scenarios. Here are some advanced techniques along with examples:

Grouping and Capturing

Grouping allows you to define subexpressions within a regex pattern. Capturing groups extract portions of the matched text for further processing.

```
import re

# Example: Matching a date pattern and capturing the year, month, and day
date_string = "2024-03-05"
pattern = r'(\d{4})-(\d{2})-(\d{2})'
match = re.match(pattern, date_string)
if match:
 print("Year:", match.group(1))
 print("Month:", match.group(2))
 print("Day:", match.group(3))
```

Non-Capturing Groups

Non-capturing groups are like regular groups but do not capture the matched text. They are useful when you need to group expressions for alternation or quantification without extracting the text.

```
import re

# Example: Matching either "foo" or "bar" but not capturing the text
text = "foobar"
pattern = r'(?:foo|bar)'
```

```
match = re.search(pattern, text)
if match:
 print("Match found:", match.group())
```

Lookahead and Lookbehind Assertions

Lookahead and lookbehind assertions allow you to assert that a pattern is or is not followed by or preceded by another pattern, without consuming the text.

```
import re

# Example: Matching digits only if they are followed by a letter
text = "123abc 456def"
pattern = r'\d(?=[a-zA-Z])'
matches = re.findall(pattern, text)
print("Matches:", matches)
```

Backreferences

Backreferences allow you to reference capturing groups within the same regex pattern. They are useful for matching repeated patterns or ensuring consistency.

```
import re

# Example: Matching repeated words using backreferences
text = "hello hello"
pattern = r'(\b\w+\b) \1'
match = re.search(pattern, text)
if match:
 print("Repeated word:", match.group())
```

Recursive Patterns

Recursive patterns allow regex to match nested structures, such as nested parentheses or HTML tags.

```
import re

# Example: Matching nested parentheses using recursive patterns
text = "(a(b(c)))"
pattern = r'\((?:[^()]|(?R))*\)'
match = re.search(pattern, text)
if match:
 print("Nested parentheses:", match.group())
```

These advanced regular expression techniques extend the capabilities of basic pattern matching in Python, allowing you to handle more complex text processing tasks effectively.

Conclusion

In conclusion, advanced regular expression techniques in Python significantly enhance text processing capabilities. By leveraging features such as grouping and capturing, non-capturing groups, lookahead and lookbehind assertions, backreferences, and recursive patterns, developers can efficiently handle complex pattern matching scenarios. These techniques enable more precise and flexible text manipulation, empowering Python programmers to tackle diverse tasks effectively and improve the robustness and efficiency of their code.

Exercises and Answers for Advanced Regular Expression Techniques

Exercises:

Grouping and Capturing:

Write a Python program that extracts the username and domain from an email address using grouping and capturing.

Non-Capturing Groups:

Create a regular expression pattern that matches either "apple" or "banana" followed by "pie" but does not capture the fruit name.

Lookahead and Lookbehind Assertions:

Write a regular expression to match words containing "cat" only if they are not followed by "fish".

Backreferences:

Develop a regex pattern to match repeated consecutive words in a text.

Recursive Patterns:

Create a regex pattern to match nested HTML tags.

Answers:

Grouping and Capturing:

```
import re

email = "john@example.com"
pattern = r'(\w+)@(\w+\.\w+)'
match = re.match(pattern, email)
if match:
 username = match.group(1)
 domain = match.group(2)
 print("Username:", username)
 print("Domain:", domain)
```

Non-Capturing Groups:

```
import re

text = "apple pie and banana pie"
pattern = r'(?:apple|banana) pie'
matches = re.findall(pattern, text)
print("Matches:", matches)
```

Lookahead and Lookbehind Assertions:

```
import re

text = "cat but not catfish"
```

```
pattern = r'\b\w+(?!fish)\b'
matches = re.findall(pattern, text)
print("Matches:", matches)
```

Backreferences:

```
import re

text = "hello hello"
pattern = r'(\b\w+\b) \1'
match = re.search(pattern, text)
if match:
 print("Repeated word:", match.group())
```

Recursive Patterns:

```
import re

html = "<div><p>Text</p></div>"
pattern = r'<(\w+)(?:[^<]|(?R))*<\/\1>'
match = re.search(pattern, html)
if match:
 print("Nested HTML tag:", match.group())
```

These exercises provide hands-on practice with advanced regular expression techniques in Python, helping reinforce your understanding of pattern matching and text manipulation.