Python Regular Expressions: Functions and Objects

PowerPoint Presentation

Slide 1: Title

Python Regular Expressions

Functions and Objects Guide

Slide 2: Introduction

Introduction

- Building on your knowledge of metacharacters and special sequence characters
- Now exploring Python's `re` module functionality
- Focus on practical application of regex patterns
- Understanding the tools available for text manipulation

Slide 3: re.compile()

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

- Converts regex pattern into a reusable regex object
- More efficient when using the same pattern multiple times
- Example:

```python
import re

# Compile a pattern
pattern = re.compile(r'\d+')

# Use the compiled pattern
result = pattern.search("I have 42 apples")
print(result.group())  # Output: 42
```

Slide 4: re.match()

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

- Attempts to match pattern at the **beginning** of string
- Returns Match object or None
- Example:

```python
# Match at the beginning
result = re.match(r'Hello', 'Hello, world!')
print(result.group())  # Output: Hello

# No match (not at beginning)
result = re.match(r'world', 'Hello, world!')
print(result)  # Output: None
```

Slide 5: re.search()

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

- Searches for the **first** occurrence of pattern anywhere in string
- Returns Match object or None
- Example:

```python
# Search anywhere in the string
result = re.search(r'world', 'Hello, world!')
print(result.group())  # Output: world
```

Slide 6: match() vs search()

`match()` **vs** `search()`

- `match()`: Pattern must be at the start of the string
- `search()`: Pattern can be anywhere in the string
- Example:

```python
text = "Hello, world!"

# Both work when pattern is at beginning
match1 = re.match(r'Hello', text)    # ✓ Works
search1 = re.search(r'Hello', text)  # ✓ Works

# Only search works when pattern is elsewhere
match2 = re.match(r'world', text)    # ✗ Returns None
search2 = re.search(r'world', text)  # ✓ Works
```

Slide 7: re.findall()

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

- Returns all non-overlapping matches as a list of strings
- Example:

```python
# Find all occurrences of digits
text = "I have 42 apples and 31 oranges"
result = re.findall(r'\d+', text)
print(result)  # Output: ['42', '31']

# Find all words
words = re.findall(r'\w+', 'Hello, world!')
print(words)  # Output: ['Hello', 'world']
```

Slide 8: re.finditer()

`re.finditer(pattern, string, flags=0)`

- Like `findall()`, but returns an iterator of Match objects
- Provides more information about each match
- Example:

python

```python
text = "I have 42 apples and 31 oranges"
matches = re.finditer(r'\d+', text)

for match in matches:
    print(f"Found '{match.group()}' at position {match.start()}-{match.end()}")
# Output:
# Found '42' at position 7-9
# Found '31' at position 20-22
```

Slide 9: re.split()

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

- Splits string by occurrences of pattern
- Returns list of substrings
- Optional maxsplit parameter limits number of splits
- Example:

python

```python
# Split by commas or spaces
text = "apple, banana  orange, grape"
result = re.split(r'[,\s]+', text)
print(result)  # ['apple', 'banana', 'orange', 'grape']

# Limit splits
result = re.split(r'[,\s]+', text, maxsplit=2)
print(result)  # ['apple', 'banana', 'orange, grape']
```

Slide 10: re.sub()

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

- Replaces occurrences of pattern with replacement string
- Optional count parameter limits number of replacements
- Example:

python

```python
# Replace digits with 'X'
text = "Phone: 123-456-7890"
result = re.sub(r'\d', 'X', text)
print(result)  # Output: Phone: XXX-XXX-XXXX

# Limit replacements
result = re.sub(r'\d', 'X', text, count=4)
print(result)  # Output: Phone: XXXX-456-7890
```

Slide 11: re.sub() with function

`re.sub()` with Function

- Replacement can be a function that receives match object
- Function must return replacement string
- Example:

python

```python
def double_digits(match):
    return str(int(match.group()) * 2)

text = "I have 42 apples"
result = re.sub(r'\d+', double_digits, text)
print(result)  # Output: I have 84 apples
```

Slide 12: re.subn()

`re.subn(pattern, repl, string, count=0, flags=0)`

- Like `sub()`, but returns tuple: (new_string, number_of_replacements)
- Example:

python

```python
# Replace and count
text = "Phone: 123-456-7890"
result = re.subn(r'\d', 'X', text)
print(result)  # Output: ('Phone: XXX-XXX-XXXX', 10)

# Limit replacements
result = re.subn(r'\d', 'X', text, count=4)
print(result)  # Output: ('Phone: XXXX-456-7890', 4)
```

Slide 13: Match Object Overview

Match Object

- Returned by `match()`, `search()`, and `finditer()`
- Contains information about the match
- Key methods:
  - `group()` - Returns matched text
  - `groups()` - Returns tuple of all groups
  - `groupdict()` - Returns dictionary of named groups
  - `start()`, `end()`, `span()` - Position information
  - `expand()` - Template substitution

Slide 14: group() Method

`group([group1, ...])`

- Returns substring matched by the pattern
- Can specify capturing groups by number or name
- Example:

```python
match = re.search(r'(\d+)-(\d+)', 'Product ID: 123-456')
print(match.group())   # Output: 123-456
print(match.group(0))  # Output: 123-456 (same as above)
print(match.group(1))  # Output: 123
print(match.group(2))  # Output: 456

# Named groups
match = re.search(r'(?P<first>\d+)-(?P<second>\d+)', 'ID: 123-456')
print(match.group('first'))   # Output: 123
```

Slide 15: groups() Method

`groups(default=None)`

- Returns tuple containing all subgroups of the match
- Optional default parameter for groups that didn't participate
- Example:

python

```python
match = re.search(r'(\d+)-(\d+)', 'Product ID: 123-456')
print(match.groups())  # Output: ('123', '456')

# With default value
match = re.search(r'(\d+)(-(\d+))?', 'Product ID: 123')
print(match.groups())            # Output: ('123', None, None)
print(match.groups(default=0))   # Output: ('123', None, 0)
```

Slide 16: groupdict() Method

`groupdict(default=None)`

- Returns dictionary of named subgroups
- Keys are group names, values are matched substrings
- Example:

python

```python
pattern = r'(?P<product>[\w\s]+): \$(?P<price>\d+(\.\d+)?)'
text = 'Item: Apple Juice: $5.99'
match = re.search(pattern, text)

print(match.groupdict())
# Output: {'product': 'Apple Juice', 'price': '5.99'}
```

Slide 17: Position Methods

Position Methods: `start()`, `end()`, `span()`

- `start([group])` - Index of start of match
- `end([group])` - Index of end of match
- `span([group])` - Tuple of (start, end) positions
- Example:

python

```python
match = re.search(r'(\w+), (\w+)', 'Hello, world!')

print(match.start())      # 0 (start of entire match)
print(match.end())        # 12 (end of entire match)
print(match.span())       # (0, 12)

print(match.start(2))     # 7 (start of 2nd group)
print(match.end(2))       # 12 (end of 2nd group)
print(match.span(2))      # (7, 12)
```

Slide 18: expand() Method

`expand(template)`

- Performs backreference substitution on template string
- Similar to how `sub()` works
- Example:

python

```python
match = re.search(r'(\w+), (\w+)', 'Hello, world!')

# Swap the groups
result = match.expand(r'\2 \1')
print(result)  # Output: world Hello
```

Slide 19: re.escape()

`re.escape(pattern)`

- Escapes all special regex characters in a string
- Useful when building patterns from user input
- Example:

```python
# Escape special characters
user_input = 'www.example.com?q=python+regex'
pattern = re.escape(user_input)
print(pattern)
# Output: www\.example\.com\?q=python\+regex

# Use in a search
text = "Visit www.example.com?q=python+regex for info"
match = re.search(pattern, text)
print(match.group())  # Output: www.example.com?q=python+regex
```

Slide 20: re.purge()

`re.purge()`

- Clears the regular expression cache
- Can improve performance in memory-constrained environments
- Example:

```python
import re

# After using many regex patterns
re.purge()  # Clear the cache
```

Slide 21: Regex Flags

Common Regex Flags

- `re.IGNORECASE` or `re.I`: Case-insensitive matching
- `re.MULTILINE` or `re.M`: `^` and `$` match start/end of each line
- `re.DOTALL` or `re.S`: `.` matches newlines too
- `re.VERBOSE` or `re.X`: Allow pattern comments and whitespace

Slide 22: Flag Examples

Using Regex Flags

python

```python
# Case-insensitive matching
result = re.search(r'python', 'PYTHON is amazing', re.IGNORECASE)
print(result.group())  # Output: PYTHON

# Multiline mode
text = "Line 1\nLine 2\nLine 3"
results = re.findall(r'^Line \d', text, re.MULTILINE)
print(results)  # Output: ['Line 1', 'Line 2', 'Line 3']

# Verbose mode
pattern = re.compile(r"""
    \d{3}  # Area code
    -      # Separator
    \d{3}  # Exchange code
    -      # Separator
    \d{4}  # Subscriber number
""", re.VERBOSE)
```

Slide 23: Practical Example - Email Extraction

Practical Example: Email Extraction

python

```python
import re

text = """
Contact us at info@example.com or support@company.org.
For sales inquiries, reach out to sales@example.com.
"""

# Find all email addresses
emails = re.findall(r'[\w\.-]+@[\w\.-]+\.\w+', text)
print(emails)
# Output: ['info@example.com', 'support@company.org', 'sales@example.com']
```

Slide 24: Practical Example - Log Parsing

Practical Example: Log Parsing

python

```python
import re

log_line = '192.168.1.1 - - [25/Mar/2021:10:15:32 +0000] "GET /index.html HTTP/1.1" 200 1234'

# Parse log line
pattern = r'(\d+\.\d+\.\d+\.\d+).*?\[(\d+/\w+/\d+:\d+:\d+:\d+).*?\] "(\w+) (.*?) HTTP/.*?" (\d+
match = re.search(pattern, log_line)

if match:
    ip, date, method, path, status, size = match.groups()
    print(f"IP: {ip}")       # 192.168.1.1
    print(f"Date: {date}")   # 25/Mar/2021:10:15:32
    print(f"Method: {method}")  # GET
    print(f"Path: {path}")      # /index.html
```

Slide 25: Practical Example - Form Validation

Practical Example: Form Validation

python

```python
import re

def validate_email(email):
    # Basic email validation
    pattern = r'^[\w\.-]+@[\w\.-]+\.\w+$'
    return bool(re.match(pattern, email))

def validate_password(password):
    # At least 8 chars, with digits, lowercase and uppercase letters
    pattern = r'^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{8,}$'
    return bool(re.match(pattern, password))

# Test validation
print(validate_email('user@example.com'))  # True
print(validate_email('invalid-email'))     # False
print(validate_password('Passw0rd'))       # True
print(validate_password('password'))       # False
```

Slide 26: Best Practices

Best Practices

- Use raw strings (`r'pattern'`) for regex patterns
- Test regex patterns incrementally
- Use online tools like regex101.com for testing and debugging
- Consider readability (use re.VERBOSE for complex patterns)
- Remember that simpler is often better
- Balance between regex and traditional string methods

Slide 27: Additional Resources

Additional Resources

- Python documentation: https://docs.python.org/3/library/re.html
- Regex testing: https://regex101.com/
- Regular Expressions Cookbook (O'Reilly)
- Practice challenges: https://regexcrossword.com/

Slide 28: Questions?

Questions?

Thank you!