Python Regular Expressions: Functions and Objects

PowerPoint Presentation with Practice Examples

Slide 1: Title

Python Regular Expressions

Functions and Objects Guide

Practice Example:
Ask students to introduce themselves and share one text processing task they'd like to solve with regular expressions. This helps you tailor examples to their interests.

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

Practice Example:

```python
import re

# Have students identify the metacharacters in these patterns
patterns = [
    r'\d+',            # One or more digits
    r'[a-zA-Z_]+',     # One or more letters or underscores
    r'^\w+@\w+\.\w+$', # Simple email pattern
    r'(\d{3})-(\d{2})-(\d{4})' # SSN format with groups
]

# Discuss as a class what text each pattern would match
```

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

  ```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
  ```

Practice Example:

python

```python
import re
import time

# Performance comparison
text = "Python was created by Guido van Rossum in the year 1991."
search_term = r'\d+'

# Method 1: Using re.compile()
start = time.time()
for _ in range(10000):
    pattern = re.compile(search_term)
    result = pattern.search(text)
end = time.time()
print(f"Using compile: {end - start:.6f} seconds")

# Method 2: Direct search
start = time.time()
for _ in range(10000):
    result = re.search(search_term, text)
end = time.time()
print(f"Direct search: {end - start:.6f} seconds")

# Have students try different patterns and text lengths
# to see when compilation becomes more efficient
```

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
  ```

Practice Example:

python

```python
import re

# Create a function to test matching different patterns
def test_match(pattern, texts):
    compiled = re.compile(pattern)
    for text in texts:
        match = compiled.match(text)
        if match:
            print(f"✓ '{text}' matches with '{match.group()}'")
        else:
            print(f"✗ '{text}' does not match")
    print()

# Practice 1: Match strings starting with 'Python'
test_match(r'Python', [
    'Python is fun',
    'I love Python',
    'python is case-sensitive',
    'Python 3.9'
])

# Practice 2: Match strings starting with a number
test_match(r'\d+', [
    '123 Main St',
    'A123 Main St',
    '123.456',
    ' 123 hidden'
])

# Challenge: Create a pattern that matches lines starting
# with a date in format YYYY-MM-DD
# Have students write and test this pattern
```

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

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

Practice Example:

python

```python
import re

# Function to highlight where matches occur in a string
def highlight_match(pattern, text):
    match = re.search(pattern, text)
    if match:
        start, end = match.span()
        result = f"{text[:start]}[{text[start:end]}]{text[end:]}"
        print(f"Match found: {result}")
        print(f"Position: {start}-{end}")
    else:
        print(f"No match in '{text}'")
    print()

# Practice 1: Search for phone numbers
pattern = r'\(\d{3}\) \d{3}-\d{4}'
texts = [
    "Call me at (555) 123-4567 tomorrow",
    "My number is 555-123-4567",
    "No phone number here"
]

for text in texts:
    highlight_match(pattern, text)

# Practice 2: Search for hashtags
hashtag = r'#\w+'
tweet = "I love #Python and #RegularExpressions are powerful! #coding"
while True:
    match = re.search(hashtag, tweet)
    if not match:
        break
    print(f"Found: {match.group()}")
    # Remove the found hashtag and continue searching
    tweet = tweet[:match.start()] + tweet[match.end():]

# Have students create patterns to search for URLs or email addresses
```

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
  ```

Practice Example:

```python
import re

def compare_match_search(pattern, strings):
    print(f"Pattern: {pattern}")
    print("-" * 40)
    print(f"{'String':<20} | {'match()':<10} | {'search()':<10}")
    print("-" * 40)

    for string in strings:
        match_result = "✓" if re.match(pattern, string) else "✗"
        search_result = "✓" if re.search(pattern, string) else "✗"
        print(f"{string:<20} | {match_result:<10} | {search_result:<10}")
    print()

# Example 1: Word boundary pattern
compare_match_search(r'\bpython\b', [
    'python',
    'python3',
    'I love python',
    'monty-python',
    'Pythonic'
])

# Example 2: Digit pattern
compare_match_search(r'\d{2,4}', [
    '123',
    'abc123',
    '1',
    'no digits'
])

# Exercise: Have students predict the results first, then verify
compare_match_search(r'[A-Z][a-z]+', [
    'Hello',
    'hello',
    'HELLO',
    'Hello world',
    'world Hello'
])
```

Slide 7: re.findall()

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

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

  python

  ```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']
  ```

Practice Example:

```python
import re

# Example 1: Extract all dates from text
text = """
Important dates:
- Project start: 2023-01-15
- First milestone: 2023-02-28
- Second milestone: 2023-04-10
- Project end: 2023-06-30
"""

dates = re.findall(r'\d{4}-\d{2}-\d{2}', text)
print("All dates found:", dates)

# Example 2: Extract words of different lengths
text = "The quick brown fox jumps over the lazy dog"

# Words with exactly 3 letters
three_letter_words = re.findall(r'\b\w{3}\b', text)
print("3-letter words:", three_letter_words)

# Words with 4 or more letters
longer_words = re.findall(r'\b\w{4,}\b', text)
print("4+ letter words:", longer_words)

# Exercise: Extract all capitalized words
text = "Python was created by Guido van Rossum. Java was created by James Gosling at Sun Micros
# Have students write the pattern
capitalized = re.findall(r'\b[A-Z][a-zA-Z]*\b', text)
print("Capitalized words:", capitalized)

# Challenge: Extract all HTML tags from an HTML snippet
html = "<div><h1>Title</h1><p>This is a <b>bold</b> paragraph with <a href='#'>link</a>.</p></d
# Have students write the pattern
```

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

```
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
```

Practice Example:

```
python
```

```python
import re

# Example 1: Analyze word positions in a sentence
text = "The quick brown fox jumps over the lazy dog"
word_matches = re.finditer(r'\b\w+\b', text)

print("Word positions:")
for i, match in enumerate(word_matches, 1):
    word = match.group()
    position = match.span()
    print(f"Word {i}: '{word}' at positions {position}")

# Example 2: Extract and analyze links from HTML
html = """
<div class="content">
  <a href="https://python.org">Python</a>
  <a href="https://docs.python.org/3/library/re.html">Regex Docs</a>
  <a href="mailto:info@example.com">Contact Us</a>
</div>
"""

# Pattern to find href attributes
href_pattern = r'href=[\'"]([^\'"]+)[\'"]'
link_matches = re.finditer(href_pattern, html)

print("\nLink analysis:")
for match in link_matches:
    full_match = match.group(0)  # The entire match (href="...")
    link = match.group(1)        # Just the URL (captured group)

    # Determine link type
    if link.startswith("http"):
        link_type = "Web URL"
    elif link.startswith("mailto:"):
        link_type = "Email"
    else:
        link_type = "Other"

    print(f"- {link_type}: {link}")
    print(f"  Full attribute: {full_match}")
    print(f"  Found at position: {match.span()}")

# Exercise: Find all color hex codes in CSS
css = """
body {
  background-color: #f0f0f0;
```

```
    color: #333;
  }
  .header {
    border-bottom: 1px solid #ccc;
    color: #0088cc;
  }
  """
  # Have students write the pattern and process with finditer
```

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']
  ```

Practice Example:

python

```python
import re

# Example 1: Compare regular split vs regex split
text = "apple,banana;orange\tgrape|cherry"

# Using standard split (only works with one delimiter)
print("Standard split by comma:", text.split(','))

# Using re.split with multiple delimiters
print("Regex split by any delimiter:", re.split(r'[,;\t|]', text))

# Example 2: Parse a semi-structured text
data = """
Name: John Smith
Age: 35
Email: john@example.com
Phone: 555-123-4567
"""

# Split by colon and trim whitespace
pairs = []
for line in data.strip().split('\n'):
    if line:  # Skip empty lines
        key_value = re.split(r':\s*', line, maxsplit=1)
        if len(key_value) == 2:
            pairs.append(key_value)

print("Parsed data:")
for key, value in pairs:
    print(f"- {key}: {value}")

# Example 3: Split by sentence boundaries
text = "Hello! This is a sample text. It has multiple sentences... Do you like it? I hope so."
sentences = re.split(r'[.!?]+\s*', text)
print("\nSentences:")
for i, sentence in enumerate(sentences, 1):
    if sentence:  # Skip empty strings
        print(f"{i}. {sentence}")

# Exercise: Parse CSV with potential quoted fields
csv_line = 'John,"Smith, Jr.",42,"New York, NY",Engineer'
# Have students write a regex that properly splits this CSV
# (Hint: splitting by commas not inside quotes)
```

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
  # 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
  ```

Practice Example:

python

```python
import re

# Example 1: Censoring sensitive information
text = "Credit card: 1234-5678-9012-3456, SSN: 123-45-6789"

# Censor credit card leaving just last 4 digits
censored = re.sub(r'(\d{4}-){3}(\d{4})', 'XXXX-XXXX-XXXX-\\2', text)
print("After credit card censoring:", censored)

# Censor SSN completely
censored = re.sub(r'\d{3}-\d{2}-\d{4}', 'XXX-XX-XXXX', censored)
print("After SSN censoring:", censored)

# Example 2: Format phone numbers consistently
phones = [
    "555-123-4567",
    "(555) 123-4567",
    "5551234567",
    "555 123 4567"
]

print("\nFormatted phone numbers:")
for phone in phones:
    # First remove all non-digit characters
    digits_only = re.sub(r'\D', '', phone)
    # Then format consistently
    if len(digits_only) == 10:
        formatted = re.sub(r'(\d{3})(\d{3})(\d{4})', '(\\1) \\2-\\3', digits_only)
        print(f"{phone} → {formatted}")
    else:
        print(f"{phone} → Invalid phone number")

# Example 3: Convert markdown headings to HTML
markdown = """
# Main Title
Some text here.
## Subtitle
More text.
### Section 1
Content.
"""

html = re.sub(r'^# (.+)$', r'<h1>\1</h1>', markdown, flags=re.MULTILINE)
html = re.sub(r'^## (.+)$', r'<h2>\1</h2>', html, flags=re.MULTILINE)
html = re.sub(r'^### (.+)$', r'<h3>\1</h3>', html, flags=re.MULTILINE)
```

```
print("\nConverted Markdown to HTML:")
print(html)

# Exercise: Clean up and standardize text
messy_text = "  Multiple    spaces   and\ttabs   between words!  "
# Have students write a regex to normalize all whitespace to single spaces
```

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
  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
  ```

Practice Example:

python

```python
import re

# Example 1: Convert temperatures from Celsius to Fahrenheit
def celsius_to_fahrenheit(match):
    celsius = float(match.group(1))
    fahrenheit = celsius * 9/5 + 32
    return f"{fahrenheit:.1f}°F"

text = "Today's temperature is 25°C, yesterday was 20°C."
converted = re.sub(r'(\d+(?:\.\d+)?)°C', celsius_to_fahrenheit, text)
print("Converted temperatures:", converted)

# Example 2: Convert camelCase to snake_case
def camel_to_snake(match):
    return f"{match.group(1)}_{match.group(2).lower()}"

code = "myVariableName = calculateTotalAmount(firstNumber, secondNumber)"
snake_case = re.sub(r'([a-z])([A-Z])', camel_to_snake, code)
print("\nCamelCase to snake_case:", snake_case)

# Example 3: Format numbers with thousand separators
def add_thousand_separators(match):
    number = match.group(0)
    # Insert commas from right to left
    result = ""
    for i in range(len(number), 0, -3):
        if i - 3 > 0:
            result = "," + number[i-3:i] + result
        else:
            result = number[0:i] + result
    return result

financial_text = "Company revenue was 1234567890 dollars, with expenses of 987654321 dollars."
formatted = re.sub(r'\d+', add_thousand_separators, financial_text)
print("\nFormatted numbers:", formatted)

# Exercise: Create a function to convert dates from MM/DD/YYYY to YYYY-MM-DD format
text_with_dates = "Event dates: 05/15/2023, 07/04/2023, and 12/31/2023"
# Have students implement the date_format_converter function
def date_format_converter(match):
    # Students should implement this function
    pass

# iso_dates = re.sub(r'(\d{2})/(\d{2})/(\d{4})', date_format_converter, text_with_dates)
```

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)
  ```

Practice Example:

python

```python
import re

# Example 1: Filter profanity and report occurrences
text = "This darn assignment is too difficult. I'm so darn frustrated with it!"
profanity_list = ["darn", "heck", "shoot"]

# Create a regex pattern from the profanity list
pattern = r'\b(' + '|'.join(profanity_list) + r')\b'
clean_text, count = re.subn(pattern, "****", text, flags=re.IGNORECASE)

print(f"Original: {text}")
print(f"Cleaned: {clean_text}")
print(f"Replaced {count} profanity occurrences")

# Example 2: Fix common typos and count them
typos = {
    "teh": "the",
    "thier": "their",
    "recieve": "receive",
    "occured": "occurred"
}

text = "Teh error occured when they recieve thier package."

total_fixes = 0
for typo, correction in typos.items():
    text, count = re.subn(r'\b' + typo + r'\b', correction, text, flags=re.IGNORECASE)
    if count > 0:
        print(f"Fixed '{typo}' → '{correction}' ({count} times)")
        total_fixes += count

print(f"\nCorrected text: {text}")
print(f"Total fixes: {total_fixes}")

# Example 3: Verify string substitutions worked correctly
def verify_phone_numbers(text):
    # Look for patterns that might be phone numbers
    pattern = r'\b\d{3}[-.\s]?\d{3}[-.\s]?\d{4}\b'

    # Try to standardize the format
    new_text, count = re.subn(pattern, lambda m: re.sub(r'[-.\s]', '-', m.group(0)), text)

    if count > 0:
        print(f"Standardized {count} phone numbers")
        print(f"Result: {new_text}")
    else:
        print("No phone numbers found to standardize")

# Test cases
verify_phone_numbers("Call me at 555.123.4567 or 555 987 6543")
verify_phone_numbers("No phone numbers here")

# Exercise: Replace HTML tags with their content and count how many were removed
html = "<p>This is a <b>bold</b> paragraph with a <a href='#'>link</a>.</p>"
# Have students write code to extract text content and count removed tags
```

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

Practice Example:

python

```python
import re

# Example: Exploring a Match object
pattern = r'(\w+)@(\w+)\.(\w+)'
text = "Contact me at john.doe@example.com for more information."

match = re.search(pattern, text)

if match:
    print("Full Match Object Demonstration")
    print("-" * 30)

    # Basic match information
    print(f"Full match: {match.group(0)}")
    print(f"At position: {match.span()}")

    # Groups
    print("\nGroups:")
    print(f"Group 1 (username): {match.group(1)}")
    print(f"Group 2 (domain): {match.group(2)}")
    print(f"Group 3 (TLD): {match.group(3)}")
    print(f"All groups: {match.groups()}")

    # Positions
    print("\nPositions:")
    for i in range(4):  # Group 0 (full match) plus 3 capturing groups
        print(f"Group {i}: starts at {match.start(i)}, ends at {match.end(i)}")

    # Rebuild in a different format
    print("\nRebuilding email:")
    new_format = match.expand(r'\1 AT \2 DOT \3')
    print(f"New format: {new_format}")

else:
    print("No match found")

# Exercise: Have students create their own regex with multiple
# capturing groups and explore the match object properties
```

Slide 14: group() Method

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

- Returns substring matched by the pattern

- Can specify capturing groups by number or name
- Example:

  python

  ```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
  ```

Practice Example:

python

```python
import re

# Example 1: Parsing a URL
url_pattern = r'(https?://)?(www\.)?([a-zA-Z0-9-]+)\.([a-zA-Z]{2,})(/[\w/.-]*)?'
urls = [
    "https://www.example.com/path/to/page.html",
    "http://subdomain.site.co.uk/search?q=regex",
    "www.python.org",
    "example.net/products"
]

print("URL Parsing:")
for url in urls:
    match = re.search(url_pattern, url)
    if match:
        print(f"\nAnalyzing: {url}")
        print(f"Full match: {match.group(0)}")
        print(f"Protocol: {match.group(1) or '(none)'}")
        print(f"www: {match.group(2) or '(none)'}")
        print(f"Domain name: {match.group(3)}")
        print(f"TLD: {match.group(4)}")
        print(f"Path: {match.group(5) or '(none)'}")

# Example 2: Named groups for contact information
contact_pattern = r'(?P<name>[A-Z][a-z]+ [A-Z][a-z]+)(?:,\s*(?P<title>[^,]+))?,\s*(?P<email>[\w

contacts = [
    "John Doe, CEO, john@example.com, 555-123-4567",
    "Jane Smith, jane.smith@company.org",
    "Bob Johnson, Manager, bob@dept.company.co.uk, 123.456.7890"
]

print("\nContact Parsing:")
for contact in contacts:
    match = re.search(contact_pattern, contact)
    if match:
        print(f"\nContact: {contact}")
        print(f"Name: {match.group('name')}")

        # Access optional groups safely
        title = match.group('title') if 'title' in match.groupdict() else "(not provided)"
        print(f"Title: {title}")

        print(f"Email: {match.group('email')}")

        phone = match.group('phone') if 'phone' in match.groupdict() else "(not provided)"
        print(f"Phone: {phone}")

# Exercise: Parse a log entry with timestamp, level, and message
log = "2023-05-15 14:23:45 [ERROR] Failed to connect to database: timeout error"
# Have students create a pattern with groups and extract the components
```

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
  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)
  ```

Practice Example:

python

```python
import re

# Example 1: Parsing different address formats with optional components
address_pattern = r'(\d+)\s+([A-Za-z\s]+?)\s*,?\s*(?:Apt\.?\s*(\w+))?,?\s*([A-Za-z\s]+),\s*(\w{

addresses = [
    "123 Main Street, Springfield, IL 62701",
    "456 Park Ave, Apt 789, New York, NY",
    "789 Broadway, Apt. 5C, Brooklyn, NY 11201",
    "1010 Elm Road, Chicago, IL"
]

print("Address Parsing with groups():")
for address in addresses:
    match = re.search(address_pattern, address)
    if match:
        # Get all groups with default value for missing ones
        number, street, apt, city, state, zipcode = match.groups(default="N/A")

        print(f"\nOriginal: {address}")
        print(f"Street Number: {number}")
        print(f"Street: {street}")
        print(f"Apartment: {apt}")
        print(f"City: {city}")
        print(f"State: {state}")
        print(f"ZIP: {zipcode}")

# Example 2: Parse optional parts of a command
command_pattern = r'(\w+)(?:\s+--(\w+)(?:=([^-]\S*))?)?(?:\
```