


TRƯỜNG: ĐH Công thương TP HCM		
KHOA: CÔNG NGHỆ THÔNG TIN	BUỔI 8A. Regular Expression và Web Crawling & Scraping	
BỘ MÔN: KHDL&TTNT		
MH: LẬP TRÌNH PYTHON		

A. MỤC TIÊU

- Làm việc với Regular Expression
- Giới thiệu cơ bản về Web Crawling & Scraping

B. DỤNG CỤ - THIẾT BỊ THÍ NGHIỆM CHO MỘT SV

STT	Chủng loại – Quy cách vật tư	Số lượng	Đơn vị	Ghi chú
1	Computer	1	1	

C. NỘI DUNG THỰC HÀNH

I. Tóm tắt lý thuyết

1. Regular Expression (Regex)

a. Định nghĩa và Ý nghĩa

Regular Expression, còn được gọi là Regex hoặc RE, là một chuỗi ký tự đặc biệt được sử dụng để biểu diễn mẫu và các quy tắc phù hợp trong các chuỗi văn bản. Regular Expression cung cấp một cách mạnh mẽ và linh hoạt để tìm kiếm, so khớp và thay thế các chuỗi dữ liệu dựa trên các mẫu được xác định trước.

Regular Expression thường được sử dụng trong các ngôn ngữ lập trình và các công cụ xử lý văn bản để thực hiện các tác vụ như kiểm tra tính hợp lệ của định dạng dữ liệu, trích xuất thông tin từ văn bản, thay thế các phần của văn bản bằng các giá trị khác, và nhiều tác vụ khác liên quan đến xử lý và phân tích văn bản.

b. Ứng dụng trong lập trình

- Xử lý dữ liệu đầu vào: Regular Expression được sử dụng để kiểm tra tính hợp lệ của dữ liệu đầu vào, chẳng hạn như email, số điện thoại, địa chỉ IP, vv.
- Trích xuất thông tin: Regular Expression được sử dụng để trích xuất các phần của văn bản hoặc dữ liệu dựa trên các mẫu được xác định trước.

- Thay thế và xử lý văn bản: Regular Expression cung cấp các phương pháp để thay thế, tách và xử lý các chuỗi văn bản theo các quy tắc nhất định.
- Tính linh hoạt và mạnh mẽ: Sử dụng Regular Expression, người lập trình có thể tìm kiếm và so khớp các chuỗi dữ liệu phức tạp một cách nhanh chóng và hiệu quả.

Việc hiểu và sử dụng Regular Expression là một kỹ năng quan trọng trong lập trình và xử lý dữ liệu, đặc biệt là khi làm việc với các tác vụ liên quan đến xử lý văn bản và dữ liệu từ nguồn khác nhau.

c. Cú pháp cơ bản của Regular Expression

Trong Regular Expression, cú pháp được sử dụng để mô tả các mẫu hoặc quy tắc phù hợp với các chuỗi văn bản. Cú pháp của Regular Expression bao gồm các ký tự đặc biệt, biểu thức, và cấu trúc phức tạp, cho phép người dùng xác định các mẫu cụ thể mà họ muốn tìm kiếm hoặc so khớp trong văn bản.

***** Các Ký tự Đặc biệt:***

- Ký tự Tiêu chuẩn: Các ký tự đại diện cho chính nó trong một chuỗi, chẳng hạn như ký tự "a", "b", "c" vv.
- Ký tự Đặc biệt: Các ký tự có ý nghĩa đặc biệt, chẳng hạn như:
 - \d: So khớp với bất kỳ chữ số nào.
 - \w: So khớp với bất kỳ ký tự chữ cái, số hoặc gạch dưới nào.
 - \s: So khớp với bất kỳ ký tự khoảng trắng nào.
 - \b: Biểu thị ranh giới từ.
 - \W: So khớp với bất kỳ ký tự nào không phải là chữ cái, số hoặc gạch dưới.
 - \D: So khớp với bất kỳ ký tự nào không phải là số.
 - \S: So khớp với bất kỳ ký tự nào không phải là khoảng trắng.
 - \B: Biểu thị một vị trí không phải là ranh giới từ.

***** Các Biểu thức thông dụng:***

- Ký tự Hoặc (|): Cho phép so khớp với một trong các mẫu được chỉ định. Ví dụ: "a|b" sẽ so khớp với "a" hoặc "b".

- Dấu Chấm (.): So khớp với bất kỳ ký tự nào (ngoại trừ ký tự xuống dòng). Ví dụ: "a.b" sẽ so khớp với "axb", "a1b", "a#b", vv.
- Khoảng ({ }): Cho phép xác định số lần lặp lại của một mẫu. Ví dụ: "a{3}" sẽ so khớp với "aaa".
- Ký tự Greedy (+, *, ?, {}): Ký tự Greedy sẽ so khớp với một hoặc nhiều lần lặp lại của mẫu trước đó. Ví dụ: "a+" sẽ so khớp với "a", "aa", "aaa", vv.

**** Các Cấu Trúc Phức Tạp**

- Nhóm (()): Cho phép xác định một nhóm các mẫu để áp dụng các biểu thức hoặc thực hiện các phép toán.
- Biểu thức điều kiện (?(...|...)): Cho phép xác định một mẫu dựa trên việc thực hiện kiểm tra điều kiện.

Cú pháp Regular Expression cung cấp một loạt các công cụ và khả năng mạnh mẽ để tìm kiếm, so khớp và xử lý văn bản một cách linh hoạt và hiệu quả. Qua sự kết hợp của các ký tự đặc biệt, biểu thức thông dụng và cấu trúc phức tạp, người dùng có thể xác định các mẫu phức tạp và thực hiện các tác vụ xử lý dữ liệu một cách chính xác và linh hoạt.

d. Sử dụng Regular Expression trong Python

Trong Python, thư viện chuẩn **re** cung cấp các công cụ và hàm để sử dụng Regular Expression một cách dễ dàng và linh hoạt. Dưới đây là một số phần chính của thư viện **re**:

**** Thư viện re:** Thư viện re là một phần của Python Standard Library và được sử dụng để thực hiện các thao tác Regular Expression.

**** Các hàm quan trọng:**

- **re.match(pattern, string, flags=0):** So khớp mẫu từ đầu chuỗi. Nếu có sự so khớp, hàm trả về một đối tượng Match; nếu không có, trả về None.
- **re.search(pattern, string, flags=0):** Tìm kiếm một sự so khớp của mẫu ở bất kỳ vị trí nào trong chuỗi. Nếu có sự so khớp, hàm trả về một đối tượng Match; nếu không có, trả về None.
- **re.findall(pattern, string, flags=0):** Tìm tất cả các chuỗi con trong chuỗi đầu vào mà phù hợp với mẫu. Kết quả được trả về dưới dạng danh sách các chuỗi.

- **re.split(pattern, string, maxsplit=0, flags=0)**: Phân tách chuỗi thành một danh sách các chuỗi con bằng cách sử dụng mẫu như một điểm phân tách.

- **re.sub(pattern, repl, string, count=0, flags=0)**: Thay thế tất cả các sự so khớp của mẫu trong chuỗi đầu vào bằng chuỗi thay thế được cung cấp.

** Flags: Có thể truyền các flags bổ sung để điều chỉnh cách hoạt động của các hàm. Ví dụ, **re.IGNORECASE** cho phép so khớp không phân biệt chữ hoa chữ thường.

e. Ví dụ và bài tập thực hành

Bài 1: Kiểm tra tính hợp lệ của địa chỉ email

Viết một chương trình Python để kiểm tra xem một chuỗi đầu vào có phải là một địa chỉ email hợp lệ hay không.

```
import re

def is_valid_email(email):
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$'
    if re.match(pattern, email):
        return True
    else:
        return False

def main():
    email = input("Nhập địa chỉ email: ")
    if is_valid_email(email):
        print(f"{email} là một địa chỉ email hợp lệ.")
    else:
        print(f"{email} không phải là một địa chỉ email hợp lệ.")

if __name__ == "__main__":
    main()
```

Bài 2: Tìm kiếm từ trong câu

Viết một chương trình Python để kiểm tra xem một từ cụ thể có xuất hiện trong một câu hay không

```

import re

def search_word(sentence, word):
    if re.search(r'\b' + re.escape(word) + r'\b', sentence,
re.IGNORECASE):
        return True
    else:
        return False

def main():
    sentence = input("Nhập câu: ")
    word = input("Nhập từ cần tìm kiếm: ")

    if search_word(sentence, word):
        print(f"Từ '{word}' được tìm thấy trong câu.")
    else:
        print(f"Từ '{word}' không được tìm thấy trong câu.")

if __name__ == "__main__":
    main()

```

Bài 3: Phân tách chuỗi văn thành các câu

Viết một chương trình Python để phân tách một đoạn văn từ file văn bản text.txt thành các câu.

```

import re

def split_into_sentences(paragraph):
    sentences = re.split(r'(?<=[.!?]) +', paragraph)
    return sentences

def main():
    # Đọc đoạn văn từ file text.txt
    with open('text.txt', 'r') as file:
        paragraph = file.read()

    sentences = split_into_sentences(paragraph)

    print("Các câu trong đoạn văn là:")

```

```

    for index, sentence in enumerate(sentences, start=1):
        print(f"{index}. {sentence}")

if __name__ == "__main__":
    main()

```

Bài 4: Thay thế các từ trong đoạn văn và lưu kết quả vào file mới

Viết một chương trình Python để thay thế một từ hoặc cụm từ cụ thể trong một đoạn văn đọc từ file văn bản text.txt, sau đó lưu kết quả vào một file mới có tên là text_replaced.txt.

```

import re

def replace_word(paragraph, old_word, new_word):
    replaced_text = re.sub(r'\b' + re.escape(old_word) + r'\b',
new_word, paragraph)
    return replaced_text

def main():
    # Đọc đoạn văn từ file text.txt
    with open('text.txt', 'r') as file:
        paragraph = file.read()

    old_word = input("Nhập từ hoặc cụm từ cần thay thế: ")
    new_word = input("Nhập từ mới để thay thế: ")

    replaced_text = replace_word(paragraph, old_word, new_word)

    # Lưu đoạn văn đã thay thế vào file text_replaced.txt
    with open('text_replaced.txt', 'w') as file:
        file.write(replaced_text)

    print("Đã thực hiện thay thế và lưu kết quả vào file
text_replaced.txt.")

if __name__ == "__main__":
    main()

```

Bài 5: Tách từ khỏi đoạn văn: Sử dụng hàm re.findall để tìm tất cả các từ trong đoạn văn. Điều này có thể giúp phân tách đoạn văn thành danh sách các từ.

Bài 6: Tìm tất cả các từ chứa một chữ cái 'a': Sử dụng hàm `re.findall` và một pattern phù hợp để tìm tất cả các từ trong đoạn văn chứa ít nhất một chữ cái 'a'.

Bài 7: Thay thế từ cụ thể trong đoạn văn: Sử dụng hàm `re.sub` để thay thế tất cả các lần xuất hiện của một từ cụ thể trong đoạn văn bằng một từ khác.

Bài 8: Tìm tất cả các dấu câu trong đoạn văn: Sử dụng hàm `re.findall` và một pattern phù hợp để tìm tất cả các dấu câu (dấu chấm, dấu phẩy, dấu chấm than, dấu chấm hỏi, dấu chấm phẩy, ...) trong đoạn văn.

Bài 9: Tìm tất cả các từ bắt đầu bằng một chữ cái in hoa: Sử dụng hàm `re.findall` và một pattern phù hợp để tìm tất cả các từ trong đoạn văn bắt đầu bằng một chữ cái in hoa.

Bài 10: Tìm tất cả các từ kết thúc bằng một số: Sử dụng hàm `re.findall` và một pattern phù hợp để tìm tất cả các từ trong đoạn văn kết thúc bằng một số.

Bài 11: Tìm tất cả các từ có độ dài ít nhất 5 ký tự: Sử dụng hàm `re.findall` và một pattern phù hợp để tìm tất cả các từ trong đoạn văn có độ dài ít nhất là 5 ký tự.

Bài 12: Tìm tất cả các từ được viết hoa trong đoạn văn: Sử dụng hàm `re.findall` và một pattern phù hợp để tìm tất cả các từ trong đoạn văn được viết hoa (tất cả các ký tự trong từ đều là chữ cái in hoa).

Bài 13: Tách các câu từ đoạn văn: Sử dụng hàm `re.split` để phân tách đoạn văn thành các câu. Điều này có thể được thực hiện bằng cách sử dụng một pattern phù hợp để tách đoạn văn tại các dấu chấm, dấu phẩy, dấu chấm than, hoặc dấu chấm hỏi.

Bài 14: Tìm tất cả các số trong đoạn văn: Sử dụng hàm `re.findall` và một pattern phù hợp để tìm tất cả các số trong đoạn văn, bao gồm cả số nguyên và số thực.

2. Web Crawling & Scraping

a. Định nghĩa và Ý nghĩa của Web Crawling & Scraping

Web crawling là quá trình tự động duyệt qua các trang web trên Internet để thu thập thông tin và cập nhật cơ sở dữ liệu của một công cụ tìm kiếm hoặc ứng dụng khác. Các trang web được duyệt qua có thể được xác định trước (như trong trường hợp của các website có cấu trúc nhất định) hoặc được lựa chọn ngẫu nhiên. Web crawling giúp tự động thu thập dữ liệu từ hàng ngàn, thậm chí hàng triệu trang web trên Internet một cách hiệu quả. Nó là quá trình cơ bản

để xây dựng cơ sở dữ liệu của các công cụ tìm kiếm, phân tích thị trường, thu thập dữ liệu cho nghiên cứu, và nhiều mục đích khác.

Web scraping là quá trình tự động trích xuất dữ liệu từ các trang web và chuyển đổi nó thành định dạng có cấu trúc như văn bản, CSV hoặc JSON để phục vụ cho mục đích phân tích hoặc lưu trữ. Web scraping cho phép thu thập dữ liệu từ các trang web một cách tự động và có thể lặp lại. Dữ liệu này sau đó có thể được sử dụng cho một loạt các mục đích, bao gồm phân tích thị trường, đánh giá cạnh tranh, thu thập thông tin về sản phẩm và dịch vụ, tạo dữ liệu cho ứng dụng di động và nhiều ứng dụng khác.

b. Một số thư viện về Web Crawling & Scraping

**** Scrapy:**

- Scrapy là một framework mạnh mẽ và linh hoạt được viết bằng Python cho việc crawl và extract dữ liệu từ các trang web.
- Nó cung cấp một cách dễ dàng để xây dựng các Spider (các module được thiết kế để thực hiện các tác vụ crawling và scraping) và cấu hình chúng để thu thập dữ liệu từ các trang web một cách tự động.
- Scrapy hỗ trợ đa luồng và có các tính năng như ghi lại và tái sử dụng các đoạn mã, giúp bạn xây dựng các ứng dụng thu thập dữ liệu phức tạp.

**** BeautifulSoup:**

Beautiful Soup là một thư viện Python cho phép bạn dễ dàng trích xuất dữ liệu từ các tài liệu HTML và XML.

Nó cung cấp các công cụ để phân tích cú pháp và điều hướng qua các cấu trúc dữ liệu phức tạp của HTML, giúp bạn thu thập thông tin một cách linh hoạt và hiệu quả.

**** lxml:**

lxml là một thư viện Python mạnh mẽ cho việc xử lý dữ liệu XML và HTML.

Nó cung cấp một API dễ sử dụng cho việc phân tích cú pháp, tạo cấu trúc dữ liệu cây, và truy vấn thông qua các phần tử và thuộc tính của tài liệu XML và HTML.

lxml có hiệu suất cao và hỗ trợ một loạt các tính năng mạnh mẽ như XPath và XSLT.

**** Requests-HTML:**

Requests-HTML là một thư viện Python dựa trên Requests và PyQuery, cho phép bạn crawl và scrape dữ liệu từ web một cách dễ dàng và tiện lợi.

Nó cung cấp các tính năng mạnh mẽ như đánh giá JavaScript, tạo các phiên bản trang web tĩnh và động, và phân tích HTML một cách linh hoạt.

**** Selenium:**

Selenium là một công cụ tự động hóa trình duyệt web, được sử dụng phổ biến trong việc kiểm thử phần mềm và tự động hóa các nhiệm vụ trên web.

Selenium có thể được sử dụng để điều khiển trình duyệt web (như Chrome, Firefox, hoặc Edge) từ Python, cho phép bạn crawl và scrape dữ liệu từ các trang web có thể được tải bằng JavaScript.