

정규표현식 및 데이터

정규 표현식 정의

- 정규 표현식(Regular Expression)은 텍스트에서 특정 패턴을 찾기 위한 문자열 처리 방식
- 복잡한 문자열 패턴을 식별하고, 처리하며, 수정하는 데 사용되는 표현 언어의 한 형태로, 다양한 텍스트 처리 작업에서 매우 유용

정규 표현식 모듈 импорт하기

- 파이썬에서 정규 표현식을 사용하기 위해 `re` 모듈을 импорт

```
import re
```

- Regular expression operations 공식 문서

re — Regular expression operations

Source code: Lib/re/ This module provides regular expression matching operations similar to those found in Perl. Both patterns

<https://docs.python.org/3/library/re.html>

기본 패턴 매칭

정규 표현식을 사용하여 텍스트에서 패턴을 찾는 기본적인 방법
문자열 속에서 숫자를 찾는 예제

```
import re #raw string text = "오늘은 2024년 5월 6일입니다." pattern =  
re.compile(r"\d+") matches = pattern.findall(text) print(matches) #  
['2024', '5', '6'] #일반문자열 pattern = re.compile("\d+") # 숫자를 찾는 정  
규 표현식 text = "There are 123 apples" matches = pattern.findall(text)  
print(matches) # ['123']
```

아래 정규화 표현식 사이트에서 패턴 탐지 여부를 확인할 수 있다.

[Debuggex: Online visual regex tester. JavaScript, Python, and PCRE.](#)



- **.** (Dot): 기본 모드에서는 줄바꿈 문자(**\n**)를 제외한 어떤 단일 문자와도 일치, DOTALL 플래그가 설정된 경우, 줄바꿈 문자를 포함한 모든 문자와 일치
- **^** (Caret): 문자열의 시작과 일치, MULTILINE 모드에서는 각 줄바꿈 후에도 일치
- **\$**: 문자열의 끝 또는 문자열 끝의 줄바꿈 바로 전과 일치, MULTILINE 모드에서는 줄바꿈 전에도 일치
- *****: 바로 앞에 있는 패턴이 0회 이상 반복되는 것과 일치
- **+**: 바로 앞에 있는 패턴이 1회 이상 반복되는 것과 일치
- **?**: 바로 앞에 있는 패턴이 0회 또는 1회 등장하는 것과 일치
- **{m}**: 바로 앞에 있는 패턴이 정확히 m회 일치
- **{m,n}**: 바로 앞에 있는 패턴이 m회에서 n회 반복되는 것과 일치
- **[]**: 문자 집합을 나타내며, **[amk]** 는 'a', 'm', 'k' 중 하나와 일치
- **|**: 선택을 의미하며 A|B는 A 또는 B 중 하나와 일치
- **()**: 괄호 안의 정규식을 그룹으로 처리
 - 파이썬 정규 표현식에서 그룹 처리를 사용하는 이유는 다양하며, 주로 데이터 추출, 조직화 및 변환 작업을 용이하게 하기 위해 사용

- `\b`: 텍스트에서 완전히 독립적인 단어를 찾고자 할 때 특히 유용
 - 단어 경계를 나타내며, 단어 문자(`\w`, 즉 `[a-zA-Z0-9_]`에 해당하는 모든 알파벳, 숫자, 밑줄)와 단어 문자가 아닌 것들(`\W`, 즉 `[^a-zA-Z0-9_]`에 해당하는 것들) 사이의 경계에 일치.
 - 이는 단어가 텍스트의 시작이나 끝, 공백, 구두점 등에 의해 명확히 구분될 때 해당 위치에서만 일치



- `\d` - 숫자와 매치, `[0-9]`와 동일한 표현식이다.
- `\D` - 숫자가 아닌 것과 매치, `[^0-9]`와 동일한 표현식이다.
- `\s` - whitespace 문자와 매치, `[\t\n\r\f\v]`와 동일한 표현식이다. 맨 앞의 빈 칸은 공백문자(space)를 의미한다.
- `\S` - whitespace 문자가 아닌 것과 매치, `[^\t\n\r\f\v]`와 동일한 표현식이다.
- `\w` - 문자+숫자(alphanumeric)와 매치, `[a-zA-Z0-9_]`와 동일한 표현식이다.
- `\W` - 문자+숫자(alphanumeric)가 아닌 문자와 매치, `[^a-zA-Z0-9_]`와 동일한 표현식이다

메타 문자 예제 코드

- `.` 메타 문자는 제외한 모든 단일 문자와 일치
- `.`을 사용해 문자열에서 첫 번째 "a" 뒤에 오는 한 문자 찾기
- 예를 들어, 정규 표현식 `a.b`는 `acb`, `a&b`, `a b` 등 `a`와 `b` 사이에 어떤 하나의 문자가 들어가는 모든 경우와 일치

```
import re
text = "abc def"
pattern = re.compile(r'a.')
match = pattern.search(text)
if match:
    print(match.group()) # 'ab' 출력
else:
    print("No match found")
```

- ***** 메타 문자는 앞의 문자가 0번 이상 반복되는 경우와 일치
- 다음은 "a" 다음에 "b"가 0번 이상 반복되고 "c"가 오는 패턴 찾기

```
import re
text = "aac"
pattern = re.compile(r'ab*c')
matches = pattern.findall(text)
print(matches) # ['ac']
```

- 대괄호 **[]** 사이에 문자를 넣어 생성하며, 대괄호 안의 어떤 문자와도 일치
- 예를 들어 **[abc]** 는 "a", "b", "c" 중 하나와 일치

```
import re # 원시 문자열을 사용하여 모음 찾는 패턴을 컴파일하고 대소문자를 구분하지 않는 옵션 설정
pattern = re.compile(r'[aeiou]', flags=re.I)
text = "Hello World"
# 컴파일된 패턴 객체를 사용하여 findall() 메서드 실행
matches = pattern.findall(text)
print(matches) # ['e', 'o', 'o']
```

기존 파일 검색에 주민번호 탐지 기능 추가

```
import os
import re
dir_path = "uploads"
all_files = os.listdir(dir_path)
txt_files = []
for file in all_files:
    if file.endswith('.txt'):
        txt_files.append(file)
# 디렉터리 내의 txt 파일을 하나씩 불러와서 오픈
for filename in txt_files:
    file_path = os.path.join(dir_path, filename)
    with open(file_path, 'r', encoding='utf-8') as f:
        lines = f.readlines()
        for index, line in enumerate(lines):
            if line.startswith("#") or line.startswith("//"):
                print(f"주석 {file_path} {index+1}라인 : 탐지 {line}")
            if re.search(r'\d{6}[-]\d{7}', line):
                print(f"주민번호 {file_path} {index+1}라인 : 탐지 {line}")
```