11장. 정규 표현식

정규식 – Regular Expression

❖ ***정규표현식이란?***

- 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식 언어이다. 문자열의 검색과 치환을 지원한다.

정규식 – Regular Expression

- 정규표현식에 사용되는 메타문자

메타문자	설 명
[ ]	대괄호는 [ ]사이의 문자들과 일치함
-	문자의 범위를 지정하는 하이픈(-)
^	부정을 나타내는 캐럿
*****	0번 이상 반복
+	1번 이상 반복
{m}	m은 반복횟수
{n,m}	m은 반복횟수, n은 최소 반복 횟수
( )	소괄호는 서브 클래스. 그룹을 만들 때 사용

정규식 – Regular Expression

- 자주 사용하는 문자 클래스

정규 표현식	설 명
\d	숫자와 매치, [0-9]와 동일한 표현식이다.
\s	space(공백)을 표현하는 문자와 매치
\w	문자+숫자와 매치, [a-zA-Z0-9]와 동일함

정규표현식 지원 – re 모듈

- 정규 표현식 활용

1. re.compile(**'[a-z]+'**) : 정규 표현식을 컴파일 한다.
2. match("**korea**") : 문자열의 시작 부분에서 정규 표현식과 일치하는 부분을 찾음

<match 객체의 주요 메서드>

메서드	기 능
group()	매치된 문자열을 돌려준다.
start()	매치된 문자열의 시작위치를 돌려준다.
end()	매치된 문자열의 끝위치를 돌려준다
span()	매치된 문자열의 (시작, 끝)에 해당하는 튜플 반환.

정규식을 사용한 문자열 검색

- 정규 표현식 활용

```
import re

pat = re.compile("[a-z]") #정규 표현식
mat = pat.match("korea")   #조사할 문자열
print(mat)
print(mat.group())
print(mat.start())
print(mat.end())
print(mat.span())

if mat:
    print('문자열 있음: ', mat.group())
else:
    print('문자열 없음')
```

정규식을 사용한 문자열 검색

- 정규 표현식 활용
  - ✓ 메타 문자 * 과 +의 차이

```
# *은 0개이상, +는 1개 이상
pat = re.compile("a*b")
mat = pat.match("b")  #aaab
# print(mat)
if mat:
    print('문자열 있음: ', mat.group())
else:
    print('문자열 없음')
```

정규식을 사용한 문자열 검색

- 유효성 검사
  - ✓ fullmatch() 함수 – 문자열 전체가 정규 표현식과 일치하는지를 찾음

```
# 전화번호 검증
# phone_pat = re.compile('010-\d{3,4}-\d{4}')
phone_pat = re.compile("010-[0-9]{3,4}-[0-9]{4}")
mat = phone_pat.fullmatch("010-12-5678")
print(bool(mat)) #False

# 한글과 전화번호 패턴 검사
name_pat = "제갈수연";
pat = re.compile("[가-힣]{2,5}")
mat = pat.fullmatch(name_pat)
print(bool(mat)) #True
```

정규식을 사용한 문자열 검색

- 유효성 검사 예제

```
# 전화번호 패턴 유효성 검사
def validate_phone_number(phone):
    """전화번호 유효성 검사 (010-XXXX-XXXX 형식)"""
    phone_pat = re.compile("010-\d{3,4}-\d{4}")
    return bool(phone_pat.fullmatch(phone))

phone_list = [
    "010-1234-5678",  # 유효
    "010-123-4567",   # 유효
    "010-12-5678",    # 무효
    "012-1234-5678",  # 무효
    "01012345678",    # 무효
    "010-1234-567"    # 무효
]

print("=== 전화번호 검증 결과 ===")
for phone in phone_list:
    print(f"{phone}: {validate_phone_number(phone)}")
```

정규식을 사용한 문자열 검색

- 유효성 검사 예제

```
# 한글이름 패턴 유효성 검사
def validate_name(user_name):
    pattern = re.compile("^[가-힣]{2,5}$")
    return bool(pattern.fullmatch(user_name))

while True:
    user_name = input("한글 이름 입력(2~5자): ")

    if validate_name(user_name):
        print(f"이름: {user_name}")
        break
    else:
        print("올바른 한글 이름이 아닙니다. 다시 입력하세요")
```

그루핑(Grouping)

- 그루핑(Grouping)

  문자열 중에서 특정 부분의 문자열만 추출하고 싶을 때 사용한다.

group(인덱스)	설 명
group(0)	매치된 전체 문자열
group(1)	첫 번째 그룹에 해당하는 문자열
group(2)	두 번째 그룹에 해당하는 문자열
group(n)	n 번째 그룹에 해당하는 문자열

그루핑(Grouping)

- 이름과 전화번호를 분리해서 추출하기
  - **표현식** – (**그룹**)
  - **출력** – group(인덱스 번호)

```
# 그룹 - 소괄호()
phone = "jang 010-1234-5678"
pat = re.compile("(\w+)\s{1,2}(010-\d{3,4}-\d{4})")
mat = pat.match(phone)
print(mat.group())
print(mat.group(1)) #jang
print(mat.group(2)) #010-1234-5678
```

그루핑(Grouping)

- **sub()를 사용한 문자 마스킹 처리**

sub(\g <그룹 인덱스>)

```
# 폰번호 뒷 4자리 마스킹 처리
pattern = re.compile("(\w+)\s{1,2}(010-\d{3,4})-\d{4}")

print(pattern.sub("\g<1>", phone)) #jang
print(pattern.sub("\g<2>-****", phone)) #010-1234-****
```

그루핑(Grouping)

- **sub()를 사용한 문자 마스킹 처리**

sub(\g <그룹 인덱스>)

```
# 주민등록번호 마스킹 처리
data = """
kim 920815-1234567
lee 031011-4123456
"""

pat = re.compile("(\d{6})[-]\d{7}")
print(pat.sub("\g<1>-*******", data))

pat2 = re.compile("(\d{6}[-]\d{1})\d{6}")
print(pat2.sub("\g<1>******", data))
```

```
kim 920815-*******
lee 031011-*******


kim 920815-1******
lee 031011-4******
```