정규 표현식
(Regular Expression)

안 화 수

정규 표현식

- 특정한 규칙을 가진 문자열의 집합을 표현하는데 사용하는 형식 언어
- Programming Language나 Text Editor 등 에서 문자열의 검색과 치환을 위한 용도로 사용
- 입력한 문자열에서 특정한 조건을 표현할 경우 일반적인 조건문으로는 다소 복잡할 수도 있지만, 정규 표현식을 이용하면 매우 간단하게 표현
- 코드가 간단한 만큼 가독성이 떨어져서 표현식을 숙지하지 않으면 이해하기 힘들다는 문제점
- 파이썬에서 정규식은 re 모듈이 제공

정규 표현식

❖ **정규표현식을 사용하지 않고 주민번호 뒷자리를 * 로 변경**

regular01.py

```
data = """
        park 800905-1049118
        kim 700905-1059119
       """
result=[]
for line in data.split('₩n'):                          # line = "park 800905-1049118"
    word_result=[]                                     # word = "park"
    for word in line.split(' '):                       # word = "800905-1049118"
        if len(word)==14 and word[:6].isdigit() and word[7:].isdigit():
            word = word[:6]+'-'+'*******'              # word = "800905-*******"
        word_result.append(word)                       # word_result=["park","800905-*******"]
    result.append(" ".join(word_result))
print('₩n'.join(result))
```

정규 표현식

❖ **정규표현식을 사용하여 주민번호 뒷자리를 * 로 변경**

regular02.py

```
import re


data = """
        park 800905-1049118
        kim 700905-1059119
        """
# 정규 표현식을 만든다.
pat = re.compile('(₩d{6})[-](₩d{7})')                    # ( ) : 정규식을 그룹으로 묶어준다.
# sub( 바꿀 문자열, 대상 문자열 )
# g<그룹명> : 정규표현식의 첫번째 그룹을 참조함  (그룹번호는  1번부터 시작함)
print(pat.sub("₩g<1>-*******", data))
# 주민번호 앞자리를 * 문자로 변경
print(pat.sub("******-₩g<2>", data))
```

정규 표현식

❖ **메타 문자(meta characters)**

➢ 메타문자란 원래 그 문자가 가진 뜻이 아닌 특별한 용도로 사용되는 문자를 의미한다.

➢ 정규 표현식에서 사용하는 메타문자의 종류

. ^ $ * + ? { } [ ] ₩ | ( )

정규 표현식

❖ 매칭 관련 메타 문자

메타 문자	설명	
.	줄 바꿈 문자를 제외한 문자와 매치	c.t는 cat, cbt 와 매칭
^	-문자열의 시작과 매칭 -[]안에서는 반대 문자열을 의미	^cat는 cat으로 시작 c[^a]t 는 cbt와는 매칭되지만 cat와는 매칭되지 않음
$	-끝나는 문자열 -[ ]안에서는 그냥 $	
[ ]	문자 집합을 의미하는데 ,나 –를 사용할 수 있음	[abc]는 a, b, c 중 하나 [0-9]는 0부터 9까지 [0-9a-zA-Z]는 영문자나 숫자
|	또는	ca|b 는 ca 또는 cb
( )	정규식을 그룹으로 묶기	
₩s	공백문자	₩t, ₩n,₩r

정규 표현식

❖ **정규식에서 문자나 문자 패턴의 반복을 표현하는 메타 문자**

반복 관련 메타 문자

메타 문자	설명	예
*	0회 이상의 반복	ca*t는 c와 t 사이에 a를 반복할 수 있는데 생략 가능
+	1회 이상의 반복	ca+t는 c와 t 사이에 a를 반복할 수 있는데 생략은 불가능
?	0이나 1회 반복	ca? 는 ct, cat 와 매칭
{m}	m회 반복을 허용	ca{2}는 a를 2회 반복
{m,n}	m회 부터 n회까지 반복을 허용	ca{2,4}는 caat, caaat, caaaat와 매칭

정규 표현식

❖ 특수 문자 : 대문자로 사용된 것은 소문자의 반대의 기능

특수문자	설명
₩₩	백 슬래시 문자
₩d	모든 숫자와 매치, [0-9]와 동일한 표현식
₩D	숫자가 아닌것과 매치, [^0-9]와 동일한 표현식
₩s	화이트 스페이스와 매칭
₩S	화이트 스페이스가 아닌 문자와 매칭
₩w	숫자 또는 문자와 매치, [a-z,A-Z,0-9]와 동일한 표현식
₩W	숫자 또는 문자가 아닌 것과 매치, [^a-z,A-Z,0-9]와 동일한 표현식

메타문자

meta.py

```
import re

# 문자 클래스 : [ ]
# [abc] : a, b, c 중 한개의 문자라도 있으면 매치
p1 = re.compile('[abc]')

print(p1.match('a'))          # 'a' 문자가 있으므로 매치
print(p1.match('before'))     # 'b' 문자가 있으므로 매치
print(p1.match('dude'))       # None

p = re.match('[abc]', 'a')
print(p)

# dot : .
# a.b : a와 b 사이에 줄바꿈 문자를 제외한 어떤 문자가 들어가도 모두 매치
p2 = re.compile('a.b')

print(p2.match('aab'))
print(p2.match('a0b'))
print(p2.match('abc'))        # None
```

메타문자

```
# 반복 : *
# ca*t : * 문자 바로 앞에 있는 a가 0번 이상 반복되면 매치
p3 = re.compile('ca*t')
print(p3.match('ct'))
print(p3.match('cat'))
print(p3.match('caaat'))

# 반복 : +
# + 는 최소 1번 이상 반복될 때 사용한다.
# * 가 반복 횟수 0회부터라면, +는 반복횟수 1회부터 이다.
# ca+t : +문자 바로 앞에 있는 a가 1번 이상 반복되면 매치
p4 = re.compile('ca+t')
print(p4.match('ca'))                # None
print(p4.match('cat'))
print(p4.match('caaat'))

# 반복 : {m}
# ca{2}t : a가 2번 반복되면 매치
p5 = re.compile('ca{2}t')
print(p5.match('cat'))               # None
print(p5.match('caat'))
```

메타문자

```
# 반복 : {m, n}
# ca{2, 5}t : a가 2~5회 반복되면 매치
p6 = re.compile('ca{2,5}t')
print(p6.match('cat'))          # None
print(p6.match('caat'))
print(p6.match('caaaaat'))

# 반복 : ?
# ? 메타문자가 의미하는 것은  {0,1}이다.
# ab?c : b가 0~1번 사용되면 매치
p7 = re.compile('ab?c')
print(p7.match('abc'))
print(p7.match('ac'))
```

매칭

❖re 모듈의 주요 함수

- complie(pattern[.flags]): 정규식 객체를 생성
- search(pattern, string[. flags])
- match(pattern, string[. flags])
- split(pattern, string[. maxsplit=0]): pattern을 기준으로 분리
- findall(pattern, string): pattern을 만족하는 모든 문자열을 추출
- sub(pattern, repl, string[, count=0]): pattern을 찾아서 repl로 치환하는데 count는 치환 횟수를 제한

❖Match 객체의 메소드

- group(): 매칭된 문자열 반환
- groups(): 매칭된 전체 그룹 문자열을 튜플 형식으로 반환
- start(): 매칭된 문자열의 시작 위치 리턴
- end(): 매칭된 문자열의 마지막 위치 리턴
- span: 매칭된 문자열의 (시작, 끝) 위치를 리턴

매칭

❖re 모듈의 match 메소드는 문자열의 시작부터 정규식에 매칭된 경우 Match 객체를 반환

❖Search 메소드는 부분적으로 일치하는 문자열로 검사

❖매칭이 되지 않은 경우에는 아무것도 리턴하지 않습니다.

❖Match 객체의 group 메소드를 호출하면 매칭이 된 문자열을 추출합니다.

```
import re
match = re.match('[0-9]', '1234')
print(match.group())
match = re.match('[0-9]', 'abc')
print(match)
match = re.match('[0-9]+', '1234')
print(match.group())
```

매칭되는 것이 없으므로 None

매칭

❖맨 앞에 공백이 오는 경우에는 ₩s를 이용해야 합니다.

```
import re
match = re.match('[0-9]+', ' 1234')
print(match)
match = re.match('₩s[0-9]+', ' 1234')
print(match)
match = re.search('[0-9]+', ' 1234')
print(match)
```

맨 앞에 공백이 있는 경우에는 매칭이 안 됨

매칭

match01.py (1/2)

```
import re

m1 = re.match('[0-9]', '1234')
print(m1.group())                    # 1 : 매치된 문자열 반환

m2 = re.match('[0-9]', 'abc')
print(m2)                            # None : 매치된 문자 없음

m3 = re.match('[0-9]+', '1234')
print(m3.group())                    # 1234 : 매치된 문자열 반환
```

매칭

match01.py (2/2)

```
# 맨 앞에 공백 있는 경우
m4 = re.match('[0-9]+', ' 1234')
print(m4)                                # None : 매치된 문자 없음

# 맨 앞에 공백이 오는 경우에는 ₩s를 이용해야 한다.
m5 = re.match('₩s[0-9]+', ' 1234')
print(m5)                                # match=' 1234'

# search() 메소드는 문자열 전체를 검색하여 정규식과 매치되는지 검사한다.
m6 = re.search('[0-9]+', ' 1234')
print(m6)                                # match=' 1234'
```

문자열 검색

❖ **정규표현식을 이용한 문자열 검색에 사용되는 함수**

```
#    함수                    기능
#-------------------------------------------------------------------------
# match()          문자열의 처음부터 정규식과 매치되는지 검사한다.
# search()         문자열 전체를 검색하여 정규식과 매치되는지 검사한다.
# findall()        정규식과 매치되는 모든 문자열을 리스트로 리턴한다.
# finditer()       정규식과 매치되는 모든 문자열을 iterator객체로 리턴한다.
```

문자열 검색

❖ **정규표현식을 이용한 문자열 검색에 사용되는 함수**

string_find.py (1/3)

```
import re

# 영문자(a~z) 정규식 생성
p = re.compile('[a-z]+')

# 1. match() 함수
# match() 함수는 정규식과 매치될 때에는 match객체를 리턴하고,
# 매치되지 않는 경우에는 None을 리턴한다.
m1 = p.match('python')              # match='python'
m2 = p.match('Python')              # None
m3 = p.match('pythoN')               # match='pytho'
m4 = p.match('pyThon')               # match='py'
m5 = p.match('3 python')             # None
print(m1)
```

문자열 검색

❖ **정규표현식을 이용한 문자열 검색에 사용되는 함수**

string_find.py (2/3)

```
# 2. search()
s1 = p.search('python')          # match='python'
s2 = p.search('Python')          # match='ython'
s3 = p.search('pythoN')          # match='pytho'
s4 = p.search('pYthon')          # match='p'
s5 = p.search('3python')         # match='python'
print(s1)

# 3. findall() 함수
result1 = p.findall('life is too short')
print(result1)                   # ['life', 'is', 'too', 'short']

result2 = p.findall('Life is tOo shorT')
print(result2)                   # ['ife', 'is', 't', 'o', 'shor']
```

문자열 검색

❖ **정규표현식을 이용한 문자열 검색에 사용되는 함수**

string_find.py (3/3)

```
# 4. finditer() 함수
result3 = p.finditer('life is too short')
print(result3)

for r in result3:
    print(r)

result4 = p.finditer('Life is tOo shorT')
for r in result4:
    print(r)
```

매칭

```
import re
p = re.compile(r'[_a-zA-Z]₩w*')
m = p.search('123 abc 123 def')
print(m.group())
m = p.findall('123 abc 123 def')
print(m)
p = re.compile('the')
print(p.findall('The cat was hungry, They were scared because of the cat'))
p = re.compile('the', re.I)
print(p.findall('The cat was hungry, They were scared because of the cat'))
```

대소문자 구분해서 찾음

매칭

❖주민번호 검사

```
import re
#주민등록번호 정규식 객체 만들기
p = re.compile('(₩d{6})-?(₩d{7})')
num = '100000-2000000'
if p.search(num) != None:
    print("올바른 주민번호 형식입니다.")
else:    print("올바 주민번호 형식이 아닙니다.")
num = '10000-2000000'
if p.search(num) != None:
    print("올바른 주민번호 형식입니다.")
else:
    print("올바른 주민번호 형식이 아닙니다.")
```

매칭

❖ **주민번호 검사**

jumin.py

```
import re
# 주민번호 정규식 객체 만들기
p = re.compile('(₩d{6})-(₩d{7})')

if p.search(num) != None:
    print('올바른 주민번호 형식입니다.')
else:
    print('올바른 주민번호 형식이 아닙니다.')
num = '100000-2000000'
print(p.search(num))          # match='100000-2000000'
if p.search(num) != None:
    print('올바른 주민번호 형식입니다.')
else:
    print('올바른 주민번호 형식이 아닙니다.')
```

문자열 치환

sub() 함수 : 문자열을 치환 해주는 함수

형식 : sub(바꿀 문자열, 대상 문자열)

sub.py (1/2)

```
import re

# 정규표현식 생성
p = re.compile('(blue|white|red)')

# blue 또는 white 또는 red 라는 문자열이 color로 변경
print(p.sub('color', 'blue socks and red shoes'))

# 출력 결과
# color socks and color shoes
```

문자열 치환

sub.py (2/2)

```
# blue 또는 white 또는 red 라는 문자열이 color로 변경 (1번만 변경됨)
print(p.sub('color', 'blue socks and red shoes', count=1))

# 출력 결과
# color socks and red shoes
```