정규표현식

- 문자열을 처리할때 특정 패턴으로 문자열을 처리하는 방법
- 정규표현식 함수
  - match : 가장 앞에서 부터 일치하는 패턴을 찾음
  - search : 문자열에서 일치하는 가장 첫번째 패턴을 찾음
  - findall : 문자열에서 일치하는 모든 패턴을 찾음
  - split : 문자열을 특정 패턴으로 나눕니다.
  - sub : 특정패턴에 맞는 문자열을 대체하기.
- pattern
- 예제
  - 중고나라
    - 0일공이삼79Oo삼영, 0일공-이삼79-Oo삼영
    - 정규표현식을 이용 => 01023790030

1. 정규표현식 함수

In [1]:
```
import re
```

In [2]:
```
s = "fast campus datascience fighting. datascience fighting. fast campus fighting."
```

In [3]:
```
# match
result1 = re.match("fast", s)
result2 = re.match("campus", s)
result1, result2
```

Out[3]: (<re.Match object; span=(0, 4), match='fast'>, None)

In [4]:
```
# search
result1 = re.search("fast", s)
result2 = re.search("campus", s)
result1, result2
```

Out[4]:
```
(<re.Match object; span=(0, 4), match='fast'>,
 <re.Match object; span=(5, 11), match='campus'>)
```

In [5]:
```
# findall
result1 = re.findall("fast", s)
result2 = re.findall("campus", s)
result1, result2
```

Out[5]:
```
(['fast', 'fast'], ['campus', 'campus'])
```

In [7]:
```
# split
re.split("campus", s)
```

Out[7]:
```
['fast ', ' datascience fighting. datascience fighting. fast ', ' fighting.']
```

In [8]:
```
# sub
re.sub("fast", "slow", s)
```

Out[8]:
```
'slow campus datascience fighting. datascience fighting. slow campus fighting.'
```

2. 패턴

- 문자
  - \d : 숫자
  - \D : 비숫자
  - \w : 숫자, 문자, _
  - \W : 숫자, 문자, _ 제외
  - \s : 공백문자
  - \S : 비공백문자
- 지정자

In [9]:
```
import string
pt = string.printable
len(pt), pt
```

Out[9]:
```
(100,
 '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c')
```

In [11]:
```
result = re.findall("\d", pt)
"".join(result)
```

Out[11]:
```
'0123456789'
```

In [12]:
```
result = re.findall("\D", pt)
"".join(result)
```

Out[12]:
```
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'
```

In [13]:
```
result = re.findall("\w", pt)
"".join(result)
```

Out[13]:
```
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ_'
```

In [14]:
```
result = re.findall("\W", pt)
"".join(result)
```

Out[14]:
```
'!"#$%&\'()*+,-./:;<=>?@[\\]^`{|}~ \t\n\r\x0b\x0c'
```

In [15]:
```
result = re.findall("\s", pt)
"".join(result)
```

Out[15]:
```
' \t\n\r\x0b\x0c'
```

In [16]:
```
result = re.findall("\S", pt)
"".join(result)
```

Out[16]:
```
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

3. 지정자

- [] : 문자의 묶음
- - : 범위
- . : 하나의 문자
- ? : 0회 또는 1회
- * : 0회 이상
- + : 1회 이상
- {m} : m 회
- {m,n} : m ~ n 회
- () : 그룹핑

In [17]:
```
pt
```

Out[17]:
```
'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~ \t\n\r\x0b\x0c'
```

In [19]:
```
re.findall("[abcd1234]", pt)
```

Out[19]:
```
['1', '2', '3', '4', 'a', 'b', 'c', 'd']
```

In [20]:
```
# - 범위
re.findall("[a-d1-4]", pt)
```

Out[20]:
```
['1', '2', '3', '4', 'a', 'b', 'c', 'd']
```

In [24]:
```
re.findall("[a-zA-Z]", pt)[:5]
```

Out[24]:
```
['a', 'b', 'c', 'd', 'e']
```

In [25]:
```
# . 문자하나
ls = ["123aab123", "a0b", "abc"]
for s in ls:
    result = re.findall("a.b", s)
    print(s, result)
```

```
123aab123 ['aab']
a0b ['a0b']
abc []
```

In [26]:
```
# ? : ?앞에 있는 패턴을 0회 또는 1회 반복
ls = ["aab", "a0b", "abc", "accb"]
for s in ls:
    # a + 모든문자 0개 또는 1개 + b
    result = re.findall("a.?b", s)
    print(s, result)
```

```
aab ['aab']
a0b ['a0b']
abc ['ab']
accb []
```

In [28]:
```
# * : 0회 이상 반복
ls = ["ac", "abc", "abbbbc", "a3bec"]
for s in ls:
    # a + b가 0개 이상 + c
    result = re.findall("ab*c", s)
    print(s, result)
```

```
ac ['ac']
abc ['abc']
abbbbc ['abbbbc']
a3bec []
```

In [29]:
```
# + : 1회 이상 반복
ls = ["ac", "abc", "abbbbc", "a3bec"]
for s in ls:
    # a + b가 1개 이상 + c
    result = re.findall("ab+c", s)
    print(s, result)
```

```
ac []
abc ['abc']
abbbbc ['abbbbc']
a3bec []
```

In [32]:
```
# {m}, {m, n}
ls = ["ac", "abc", "abbbbc", "abbbbbbbbbbc"]
for s in ls:
    result = re.findall("ab{4}c", s)
    print(s, result)
```

```
ac []
abc []
abbbbc ['abbbbc']
abbbbbbbbbbc []
```

In [33]:
```
# () 그룹핑
ls = ["aaa5.djfi", "abdddc5", "1abbbc", "a3.bec"]
for s in ls:
    result = re.findall("([0-9]+)[.]([a-z]{2})", s)
    print(s, result)
```

```
aaa5.djfi [('5', 'dj')]
abdddc5 []
1abbbc []
a3.bec [('3', 'be')]
```

In [34]:
```
# 이메일 주소 찾기
s = "저의 이메일 주소는 pdj1224@gmail.com 이고 pythontest@daum.net도 사용합니다"
```

In [36]:
```
p = "[\w]+@[0-9a-z]+\.[0-9a-z]+"
re.findall(p, s)
```

Out[36]:
```
['pdj1224@gmail.com', 'pythontest@daum.net']
```

In [42]:
```
# 주민등록번호 : 661231-1987123 -> 661231-*******
s = "저의 주민등록 번호는 661231-1987123 입니다."
p = "([0-9]{6})[-]?([0-9]{7})"
print(re.findall(p, s))
re.sub(p, "\g<1>-*******", s)
```

```
[('661231', '1987123')]
```

Out[42]:
```
'저의 주민등록 번호는 661231-******* 입니다.'
```