15강

문자열 응용과 정규식

15-1 문자열 응용

- 문자 변환
  - ✓ translate 함수는 문자열 안의 문자를 다른 문자로 변환시켜준다.
  - ✓ str.maketrans('바꿀 문자', '새 문자')로 변환 테이블을 생성한 후 translate(테이블)을 사용하여 문자를 바꿀 수 있다.

```
table = str.maketrans('aeiou', '12345')
str1 = 'apple'.translate(table)
print(str1)
```

[결과]

1ppl2

15-1 문자열 응용

- 문자열 연결
  - ✓ join(리스트)는 리스트의 요소를 하나의 문자열로 만들어 리턴한다.
  - ✓ ' '에는 리스트의 요소를 합칠 때 삽입할 문자나 문자열을 넣어 준다.

```
data = ['red','blue','green','yellow']
str2 = ''.join(data)
print(str2)
str3 = '-'.join(data)
print(str3)
```

[결과]

```
redbluegreenyellow
red-blue-green-yellow
```

15-1 문자열 응용

- 특정 문자 삭제
  - ✓ lstrip()은 문자열에서 왼쪽에 있는 연속된 모든 공백을 삭제한다.
  - ✓ rstrip()은 문자열에서 오른쪽에 있는 연속된 모든 공백을 삭제한다.
  - ✓ strip()은 문자열에서 양쪽에 있는 연속된 모든 공백을 삭제한다.
  - ✓ lstrip('삭제할 문자들')과 같이 삭제할 문자들을 문자열 형태로 넣어주면 문자열 왼쪽에 있는 해당 문자를 삭제한다.

```
str4 = '    hello python    '
print(str4.lstrip())
print(str4.rstrip())
print(str4.rstrip())
```

```
[결과]
hello python
    hello python
hello python
```

```
str5 = ',.hello world,.'
print(str5.lstrip(',.'))
print(str5.rstrip(',.'))
print(str5.strip(',.'))
```

```
[결과]
hello world,.
,.hello world
hello world
```

15-1 문자열 응용

- 문자열 정렬
  - ✓ ljust(), rjust()는 문자열을 지정된 길이로 만든 뒤 각각 왼쪽, 오른쪽으로 정렬하며 남는 공간을 공백으로 채운다.
  - ✓ center()는 문자열을 지정된 길이로 만든 뒤 중앙으로 정렬하며 남는 공간을 공백으로 채운다.
  - ✓ zfill()는 지정된 길이에 맞춰서 문자열의 왼쪽에 0을 채운다.

```
sdata = 'good day'
print(sdata.ljust(15))
print(sdata.rjust(15))
print(sdata.center(15))
```

```
[결과]
good day
        good day
    good day
```

```
sdata2 = '432'
print(sdata2.zfill(5))
print('53.56'.zfill(7))
print('string'.zfill(10))
```

```
[결과]
00432
0053.56
0000string
```

15-2 정규 표현식

- 정규 표현식 개요
  - ✓ 정규 표현식이라는 이름은 정규 문자열을 식별하는 데서 유래되었다.
  - ✓ 정규 표현식은 문자열이 주어진 규칙에 일치하는지, 일치하지 않는지 판단할 수 있다.
    - 글자 a를 최소한 한 번 쓰시오.
    - 그 뒤에 b를 정확히 다섯 개 쓰시오.
    - 그 뒤에 c를 짝수 번 쓰시오.
    - 마지막에 d가 있어도 되고 없어도 됩니다.
    - => aaaabbbbbccccd, aabbbbbcc 등이 성립한다.
    - aa*bbbbb(cc)*(d|)

15-2 정규 표현식

- 정규 표현식 개요
  - ✓ 규칙1 : 이메일 주소의 첫 번째 부부에는 다음 중 최소한 하나가 포함 되어야합니다.
    대문자, 소문자, 숫자 0-9, 마침표(.), 플러스 기호(+), 밑 줄기호
    [A-Za-z0-9₩._+]+
  - ✓ 규칙2:그 다음에 @가 나타나야 한다.
    @
  - ✓ 규칙3:그 다음에는 반드시 대문자나 소문자가 최소한 하나 있어야 한다.
    [A-Za-z]+
  - ✓ 규칙4: 그 다음에는 마침표가 온다.
    ₩.
  - ✓ 규칙5: 이메일 주소는 com, org, edu, net중 하나로 끝난다.
    (com|org|edu|net)
  - ✓ [A-Za-z0-9₩._+]+@[A-Za-z]+₩.(com| org| edu| net)

15-2 정규 표현식

정규 표현식 기호

기호	의미	예제	일치하는 문자열 예제
*	바로 앞에 있는 문자. 대괄호로 묶인 문자들이 0번 이상 나타납니다.	a*b	aaaaaa, aaabbbbb, bbbbbbb
+	바로 앞에 있는 문자. 대괄호로 묶인 문자들이 1번 이상 나타납니다.	a+b+	aaaaaab, aabbbb, abbbbbb
[]	대괄호 안에 있는 문자 중 하나가 나타납니다.	[A-Z]*	APPLE, CAPITALS
{m,n}	m번 이상 n번 이하 문자들이 나타납니다.	A{2,3}b{2,3}	aabbb, aaabbb, aabb
[^]	대괄호 안에 있는 문자를 제외한 문자가 나타납니다.	[^A-Z]	apple, banana
|	표현식 중 하나가 나타납니다.	b(a|i|e)d	bad, bid, bed
.	문자 하나가 나타납니다.	b.d	bad, bzd, b$d
^	바로 뒤에 있는 문자 혹은 하위 표현식이 문자열 맨 앞에 나타납니다.	^a	apple, asdf, a
₩	특수 문자를 원래 의미대로 쓰게 하는 문자입니다.	₩. ₩| ₩₩	. | ₩
$	하위 표현식이 마지막이라는 뜻입니다.	[A-Z]*[a-z]*$	ABCabc, zzzyx, Bob
?!	'포함하지 않는다'는 뜻입니다.	^((?![A-Z]).)*$	No-caps-here

15-3 정규 표현식 활용1

- 정규 표현식 함수
  - ✓ 파이썬에서 정규표현식은 re 모듈을 사용한다.
  - ✓ compile 함수는 패턴 문자열을 만들어 준다. 대신해서 r' '으로 사용해도 된다.
  - ✓ match 함수는 정규표현식 패턴과 매칭되는 정보를 리턴 해준다.

```
import re

rc = re.compile('hello')
rdata = re.match('hello', 'hello, world')
rdata = re.match(r'hello', 'hello,
world')
rdata = re.match(rc, 'hello, world')
print(rdata)
print(rdata.span()[0])
print(rdata.span()[1])
```

```
[결과]
<re.Match object; span=(0, 5),
match='hello'>
0
5
```

15-3 정규 표현식 활용1

- 정규 표현식 함수
  - ✓ search 함수는 match 함수와 달리 문자열 중간 탐색도 가능하다.

```
robj2 = re.search('^hello','hello, world')
print(robj2)
robj3 = re.search('world$', 'hello world')
print(robj3)
robj4 = re.match('world','hello world')
print(robj4)
```

```
[결과]
<re.Match object; span=(0, 5), match='hello'>
<re.Match object; span=(6, 11),
match='world'>
None
```

  - ✓ 여러 문자열이 하나라도 포함 되는지 판단할 경우에는 | 연산자를 사용하면 된다.

```
robj5 = re.search('hello|world', 'good world')
print(robj5)
```

```
[결과]
<re.Match object; span=(5, 10),
match='world'>
```

15-4 정규 표현식 활용2

- 정규표현식 표현
  - ✓ [ ](대괄호) 안에 찾고자 하는 문자들을 넣는다.
  - ✓ 숫자 범위는 0-9처럼 표현하며 *는 문자(숫자)가 0개 이상 있는지, +는 1개 이상 있는지 판단한다.

```
print(re.match(r'[0-9]*', '121hello'))
print(re.match(r'[0-9]+', '121hello'))
```

```
[결과]
<re.Match object; span=(0, 3), match='121'>
<re.Match object; span=(0, 3), match='121'>
```

  - ✓ *와 + 활용은 아래 코드를 통해 확인해 볼 수 있다.

```
print(re.match(r'a*b', 'b'))
print(re.match(r'a+b', 'b'))
print(re.match(r'a*b', 'aab'))
print(re.match(r'a+b', 'aab'))
```

```
[결과]
<re.Match object; span=(0, 1), match='b'>
None
<re.Match object; span=(0, 3), match='aab'>
<re.Match object; span=(0, 3), match='aab'>
```

15-4 정규 표현식 활용2

- 정규표현식 표현
  - ✓ ?는 ? 앞의 문자(범위)가 0개 또는 1개인지 판단하고, .은 해당 위치에 아무 문자(숫자)가 1개 있는지 판단한다.

```
print(re.match(r'ab?d', 'abd'))
print(re.match(r'ab.d', 'abcd'))
```

```
[결과]
<re.Match object; span=(0, 3), match='abd'>
<re.Match object; span=(0, 4), match='abcd'>
```

  - ✓ {시작개수,끝개수}는 시작 개수와 끝 개수를 하여 범위 안에 문자열이 있는지 판단한다.

```
print(re.match(r'[0-9]{2,3}-[0-9]{3,4}-[0-9]{4}', '010-7777-7878'))
print(re.match(r'[0-9]{2,3}-[0-9]{3,4}-[0-9]{4}', '02-111-7878'))
```

```
[결과]
<re.Match object; span=(0, 13), match='010-7777-7878'>
<re.Match object; span=(0, 11), match='02-111-7878'>
```

15-4 정규 표현식 활용2

- 정규표현식 표현
  - ✓ 영문 문자 범위는 []안에 a-z, A-Z와 같이 표현해주면 된다.
  - ✓ 한글을 검색하고 싶으면 []안에 가- 와 같이 표현해주면 된다.

```
print(re.match(r'[a-zA-Z0-9]+', 'everyone5656'))
print(re.match(r'[가-  ]+', '안녕하세요5656'))
```

```
[결과]
<re.Match object; span=(0, 12),
match='everyone5656'>
<re.Match object; span=(0, 5), match='안녕하세요'>
```

  - ✓ 특정 문자 범위에 포함되지 않는지 판단하려면 []범위 앞에 ^를 사용하면 된다.

```
print(re.match(r'[^A-Z]+', 'everyONE'))
print(re.match(r'[^0-9]+', '안녕하세요5656'))
```

```
[결과]
<re.Match object; span=(0, 5), match='every'>
<re.Match object; span=(0, 5), match='안녕하세요'>
```

15-4 정규 표현식 활용2

- 정규표현식 표현
  - ✓ ₩d, ₩D, ₩w, ₩W를 사용하면 숫자와 문자를 판단할 때는 편리하게 사용할 수 있다.

```
print(re.match(r'₩d+', '342everyone'))
print(re.match(r'₩D+', 'everyone5656'))
print(re.match(r'₩w+', '342everyone'))
print(re.match(r'₩W+', '()everyone5656'))
```

```
[결과]
<re.Match object; span=(0, 3), match='342'>
<re.Match object; span=(0, 8), match='everyone'>
<re.Match object; span=(0, 11),
match='342everyone'>
<re.Match object; span=(0, 2), match='()'>
```

  - ✓ ()를 이용하여 그룹 지정하여 사용할 수 있으며, 해당 그룹과 일치하는 문자열을 얻어 올 수 있다.

```
m = re.match('([0-9]+) ([0-9]+)', '10 2283')
print(m.group(1))
print(m.group(2))
print(m.group(0))
print(m.group())
print(m.groups())
```

```
[결과]
10
2283
10 2283
10 2283
('10', '2283')
```

15-4 정규 표현식 활용2

- 정규표현식 표현
  - ✓ findall 함수를 사용하며 패턴에 매칭되는 모든 문자열을 가져올 수 있다.

```
fdata = re.findall('[0-9]+', '12 22 4 string python 3 82')
print(fdata)
```

[결과]

['12', '22', '4', '3', '82']

  - ✓ sub 함수는 기존 문자열을 특정 문자열로 치환할 때 사용한다.

```
sdata = re.sub('apple|orange', 'fruit', 'apple box orange tree')
print(sdata)
sdata2 = re.sub('[0-9]+', 'num', '12 22 4 string python 3 82')
print(sdata2)
```

[결과]

fruit box fruit tree

num num num string python num num

15-4 정규 표현식 활용2

- 정규표현식 표현
  - ✓ sub 함수는 바꿀 문자열 대신 함수를 이용하여 지정할 수도 있다.
  - ✓ 이때 사용되는 함수는 매개변수로 매치 객체를 받으며 바꿀 결과를 문자열로 반환 시켜주면 된다.
  - ✓ 람다표현식을 이용하면 코드가 더 간결하게 만들 수 있다.

```
def multiple(m):
    n = int(m.group())
    return str(n*10)

sdata3 = re.sub('[0-9]+', multiple, '12 22 4 string python 3 82')
print(sdata3)

sdata4 = re.sub('[0-9]+', lambda m:str(int(m.group())*10), '12 22 4 string python 3 82')
print(sdata4)
```

```
[결과]
120 220 40 string python 30 820
120 220 40 string python 30 820
```