Python
CH08: 텍스트 함수

전종준 교수

서울시립대학교 통계학과/통계데이터사이언스대학원

Contents

- 문자열의 처리

- 문자열의 검색

문자열의 처리

문자열 기본 개념

- 문자열은 작은따옴표나 큰따옴표로 감쌉니다.
- 여러 줄 문자열은 세 개의 따옴표 사용

```
1 s1 = 'Hello'
2 s2 = "World"
3 s3 = '''여러 줄
4 문자열입니다.'''
```

문자열 인덱싱과 슬라이싱

- 문자열은 시퀀스형 → 인덱싱 가능
- 부분 문자열 추출은 슬라이싱

```
1  text = "Hello"
2  print(text[0])      # H
3  print(text[-1])     # o
4  print(text[1:4])    # ell
```

문자열 연산의 기본

- + : 문자열 이어 붙이기
- * : 문자열 반복
- += : 문자열 누적 결합

```
1 msg = "Hi"
2 msg += "!"
3 print(msg * 3)     # Hi!Hi!Hi!
```

문자열 연산 시 유의사항

- 문자열은 **불변(immutable)** → 수정 불가, 새로운 문자열 생성
- 숫자와 문자열은 직접 결합 불가 → `str()`로 변환 필요
- 반복적인 문자열 결합 시 `''.join()` 사용이 성능상 유리

```
# 형변환 필요
age = 10
print("나이: " + str(age))

# join을 이용한 효율적인 문자열 결합
words = ["Python", "is", "fun"]
print(" ".join(words))
```

문자열 길이 구하기

- 문자열의 길이는 `len()` 함수로 측정
- 공백, 특수문자도 모두 1글자로 계산

```
1 msg = "안녕하세요!"
2 print(len(msg))  # 6
3 print(len(" "))   # 1
```

대소문자 변환

- `lower()` : 모두 소문자로
- `upper()` : 모두 대문자로
- `capitalize()` : 첫 글자만 대문자
- `title()` : 각 단어의 첫 글자만 대문자

```
1 s = "python programming"
2 print(s.upper())        # PYTHON PROGRAMMING
3 print(s.title())        # Python Programming
```

공백 및 특정 문자 제거

- `strip()` : 양쪽 공백 제거
- `lstrip()`, `rstrip()` : 왼쪽/오른쪽 제거
- `replace(old, new)` : 특정 문자 바꾸기

```
1 text = "  Hello!  "
2 print(text.strip())              # "Hello!"
3 print(text.replace("!", "."))    # "  Hello.  "
```

문자열 나누기와 합치기

- `split()` : 문자열을 리스트로 분할
- `join()` : 리스트를 구분자 기준으로 결합

```
1 s = "apple,banana,grape"
2 fruits = s.split(",")            # ['apple', 'banana', 'grape']
3 joined = "|".join(fruits)        # apple|banana|grape
```

문자열 검색 함수

- `find()` : 처음 등장하는 위치 반환 (없으면 -1)
- `index()` : find와 같지만, 없으면 오류 발생
- `count()` : 특정 문자열의 등장 횟수

```
1 word = "banana"
2 print(word.find("a"))    # 1
3 print(word.count("a"))   # 3
```

문자열 시작과 끝 확인

- `startswith("문자열")` : 지정한 문자열로 시작하는가?
- `endswith("문자열")` : 지정한 문자열로 끝나는가?
- 확장자 검사, 파일명 필터링 등에 유용

```
1 filename = "lecture.pdf"
2 print(filename.endswith(".pdf"))       # True
3 print(filename.startswith("lec"))      # True
```

실습: 이메일 파싱

- 이메일에서 사용자명과 도메인 추출

```
email = "user@example.com"
user, domain = email.split("@")
```

연습문제 1: 이메일 정보 추출

- 사용자로부터 이메일 주소를 입력받아
- 아이디와 도메인을 각각 출력하는 프로그램을 작성하세요.

예시 입력:

```
pythonuser@example.com
```

예시 출력:

```
아이디: pythonuser
도메인: example.com
```

연습문제 2: 단어 개수 세기

- 사용자로부터 문장을 입력받아
- 단어의 개수를 출력하는 프로그램을 작성하세요.
- 단어는 공백(" ")을 기준으로 구분합니다.

예시 입력:

```
Python is fun and powerful
```

예시 출력:

```
단어 개수: 5
```

문자열의 검색

문자열 포함 조건문

- `in`, `not in` 활용

```
1 text = "I love python"
2 print("python" in text)
```

실습: 금지어 필터링

- 금지어 리스트 검사

```
text = "너무 나빠!"
for bad in ["나빠", "싫어"]:
    if bad in text:
        print("금지어 포함!")
```

replace() 함수와 텍스트 정제

- replace(old, new)는 문자열 내의 특정 문자를 다른 문자로 변경
- 여러 개의 불필요한 문자 제거 시 replace()를 반복 사용하거나 re.sub()로 대체할 수 있음
- 텍스트 정제(cleaning) 시 자주 사용: 기호 제거, 단어 치환 등

```
txt = " Hello, World!!! "
clean = txt.strip()              # 양쪽 공백 제거
clean = clean.replace("!", "")   # 느낌표 제거
clean = clean.replace(",", "")   # 쉼표 제거
clean = clean.lower()            # 모두 소문자
print(clean)                     # hello world
```

replace() 반복 줄이기: re.sub() 활용

- replace()를 여러 번 호출하는 대신, re.sub()로 한 번에 처리할 수 있음
- [!,.]는 느낌표, 쉼표, 마침표 중 하나에 해당하는 문자 패턴

```python
import re

text = " Hello, World!!! "
clean = text.strip().lower()
clean = re.sub(r"[!,.]", "", clean)
print(clean)    # hello world
```

replace()만으로 다양한 형태 치환하기

- 아래 문장에서 "스팸" 관련 표현을 모두 제거하고 싶다고 가정
- replace()로는 모든 경우를 일일이 처리해야 함

```
1 text = "SPAM is bad. spam! Don't eat s.p.a.m or Spam?"
2 text = text.replace("SPAM", "")
3 text = text.replace("spam!", "")
4 text = text.replace("Spam?", "")
5 text = text.replace("s.p.a.m", "")
6 print(text)
```

문제점: 표현이 늘어날수록 코드가 복잡하고 유지보수가 어려워짐

정규표현식으로 다양한 표현 한 줄에 처리하기

- `re.sub()`를 사용하면 유사 표현을 한 줄로 간결하게 제거할 수 있어요.
- 정규식 `[\W_]`는 알파벳/숫자가 아닌 문자(기호, 공백, 점 등)를 의미합니다.
- `flags=re.I`를 사용하면 대소문자를 구분하지 않습니다.

```
1 import re
2
3 text = "SPAM is bad. spam! Don't eat s.p.a.m or Spam?"
4 clean = re.sub(r"s[\W_]*p[\W_]*a[\W_]*m[\W_]*", "", text, flags=re.I)
5 print(clean)
```

장점: 다양한 표현(SPAM, spam!, s.p.a.m, Spam?)을 한 줄로 제거할 수 있습니다.

정규표현식이란?

- 특정 패턴의 문자열을 찾는 도구
- `import re`로 사용
- 기본 함수:
  - `re.search()` : 문자열 전체에서 패턴이 있는지 찾음
  - `re.match()` : 문자열의 시작에서 패턴이 있는지 확인
  - `re.findall()` : 패턴에 일치하는 모든 결과를 리스트로 반환

```
1  import re
2
3  result = re.search("apple", "I like apple pie")
4  print(result)
```

re.search() 예시

- 문자열 안에서 특정 단어가 포함되어 있는지 확인할 때 사용
- 함수는 일치하는 첫 번째 결과를 찾고, 결과가 있으면 Match 객체를 반환함

`group()`	일치한 문자열 자체 반환
`span()`	(시작 인덱스, 끝 인덱스) 튜플 반환
`start()`	일치한 문자열의 시작 위치
`end()`	일치한 문자열의 끝 위치

```
1 import re
2 text = "I like apple pie"
3 result = re.search("apple", text)
4 print(result)      # <re.Match object; span=(7, 12), match='apple'>
5 print(bool(result))      # True
6 print(result.span())   # (7, 12)
7 print(result.group())  # 'apple'
8 print(result.start())   # 7
9 print(result.end())     # 12
```

re.match() 예시

- 문자열이 특정 단어로 시작하는지 확인하고 Match 객체를 반환

```python
import re

text = "Hello world"
result = re.match("Hello", text)
print(result)          # <re.Match object; span=(0, 5), match='Hello'>
print(result.group())   # 'Hello'
print(result.span())    # (0, 5)
print(result.start())   # 0
print(result.end())     # 5

result2 = re.match("world", text)
print(result2)         # None (처음이 아니기 때문)
```

re.findall() 예시

- 일치하는 모든 항목을 리스트로 반환
- `Match` 객체를 반환하지 않으며, `group()`, `span()` 등을 사용할 수 없음
- 일치 항목이 없으면 빈 리스트 `[]`를 반환

```
1  import re
2
3  text = "I have an apple and an orange"
4  result = re.findall("an", text)
5  print(result)     # ['an', 'an']
```

실습 1: 단어 포함 여부 확인 (re.search)

- 문자열에 "banana"가 포함되어 있는지 확인
- re.search()를 사용

예시 코드:

```
import re

text = "I bought apple and banana today."
result = re.search("banana", text)

if result:
    print("찾았습니다!")
else:
    print("없어요.")
```

실습 2: 문자열 시작 확인 (re.match)

- 문자열이 `"Hello"`로 시작하는지 확인해 보세요.
- `re.match()`를 사용하세요.

예시 코드:

```
import re

text = "Hello, how are you?"
result = re.match("Hello", text)

if result:
    print("문장이 'Hello'로 시작합니다.")
else:
    print("시작하지 않습니다.")
```

실습 3: 단어 개수 세기 (re.findall)

- 문장에 "dog"라는 단어가 몇 번 등장하는지 세어 보세요.
- `re.findall()`을 사용하세요.

예시 코드:

```
1 import re
2
3 text = "My dog is cute. Another dog is big. Dog dog dog."
4 result = re.findall("dog", text)
5
6 print("dog는 총", len(result), "번 등장했습니다.")
```

기본 메타 문자

- `.` : 임의의 한 문자
- `^` : 문자열의 시작
- `$` : 문자열의 끝
- `[]` : 문자 집합

메타 문자 예시

```
1  import re
2
3  print(re.search("a.b", "acb"))      # 매치됨
4  print(re.search("a.b", "a\nb"))     # 매치 안됨 (\n은 .에 포함되지 않음)
5  print(re.search("^Hello", "Hello World"))  # 시작 확인
6  print(re.search("World$", "Hello World"))  # 끝 확인
```

반복을 나타내는 방법: {n}

- 특정 문자가 n번 반복되는 경우를 표현할 수 있음
- a{3}는 "aaa"에 일치
- {n,m}는 반복 횟수를 범위로 지정할 수 있음
- a{2,4} : "a"가 **2번 이상 4번 이하** 반복될 때 일치

예시 코드:

```
1  import re
2
3  print(re.search("a{3}", "aaabbb"))    # 'aaa'에 일치
4  print(re.search("a{2}", "abcabc"))  # 'aa' 없음 → None
5  print(re.search("a{2,4}", "a"))       # None
6  print(re.search("a{2,4}", "aa"))      # 'aa'
7  print(re.search("a{2,4}", "aaaaa"))   # 'aaaa' (최대 4개까지만 일치), 가장 많이 일치하는 부
8  print(re.search("a{2,}", "aaaaaaa"))  # 'aaaaaaa'
```

반복 메타 문자: *, +, ?

- 이 메타문자들은 **바로 앞에 있는 문자 또는 그룹**에 적용됩니다.
- `a*` : "a"가 0번 이상 반복, `a{0,}` 와 같음 (0회 이상 반복)
- `a+` : "a"가 1번 이상 반복, `a{1,}` 와 같음 (1회 이상 반복)
- `a?` : "a"가 0번 또는 1번만 나옴, `a{0,1}` 와 같음 (0회 또는 1회)

예시 코드:

```
1 print(re.search("a*", "aaabb"))    # 'aaa'
2 print(re.search("a+", "aaabb"))    # 'aaa'
3 print(re.search("a?", "bbbbb"))    # '' (0번도 가능)
```

그룹과 선택

- ( ) : 소괄호 ( )는 하나의 **묶음(그룹)**을 만들 때 사용
- ( ) 는 반복이나 선택을 **하나의 단위**로 적용할 수 있음. 예) (ab){2} 는 ab가 두 번 반복되는 것.
- | : |는 **"또는"**을 의미
- texttta|b는 "a 또는 b"와 일치
- 괄호와 함께 쓰면 **문자열 조각 단위** 선택도 가능. 예를 들면 (dog|cat) 은 dog 또는 cat 을 의미함.

```
1  import re
2
3  print(re.search("gr(a|e)y", "gray"))    # 매치됨
4  print(re.search("gr(a|e)y", "grey"))    # 매치됨
5
6  text = "I like orange juice."
7  result = re.search("apple|banana|orange", text)
8  print(result.group())  # 'orange'
```

그룹과 반복을 결합해서 사용하기

- `(ab)+` : 'ab'라는 패턴이 반복되는 경우
- `(go|come)2` : 'go' 또는 'come'이 2번 반복

```
1 print(re.search("(ab)+", "ababb"))        # 'abab'
2 print(re.search("(go|come){2}", "gocomecome"))  # 'comecome'
```

문자 집합: [ ] 대괄호

- [ ]는 여러 문자 중 하나를 선택할 수 있는 **문자 집합**입니다.
- [abc] → 'a', 'b', 'c' 중 하나와 일치
- [0-9] → 숫자 하나와 일치 (0 9)
- [^abc] → 'a', 'b', 'c'가 아닌 문자에 일치

예시 코드:

```python
import re

print(re.search("[aeiou]", "python"))      # 'o'
print(re.search("[0-9]", "abc123"))        # '1'
print(re.search("[^0-9]", "123a456"))      # 'a'
print(re.findall("[A-Z]", "Hello World")) # ['H', 'W']
print(re.findall("[가-힣]", "안녕 World"))
```

이스케이프 문자란?

- 백슬래시 (\)로 시작하는 특수 문자
- **화면에 출력하기 어려운 문자**를 표현할 때 사용
- 문자열 내에서 특별한 동작을 하도록 만들어 줌

자주 쓰는 이스케이프 문자:

- `\n` : 줄바꿈 (newline)
- `\t` : 탭 (tab)
- `\"` : 큰따옴표 (")
- `\\` : 백슬래시 자체 (\)

예시 코드:

```
1 print("안녕하세요\n반갑습니다")  # 줄바꿈 포함
2 print("경로: C:\\Users\\Name")  # 백슬래시 출력
```

파이썬의 raw string (r"...")

- r"" 형태는 **이스케이프 문자를 그대로 보존**
- 정규표현식에서 \를 많이 쓰기 때문에 자주 사용
- 예: r"\d{3}-\d{4}"는 숫자 3자리-4자리 패턴

예시 코드:

```
1  import re
2
3  pattern1 = "\\d{3}-\\d{4}"      # 일반 문자열
4  pattern2 = r"\d{3}-\d{4}"       # raw 문자열
5
6  print(re.search(pattern1, "123-4567"))  # OK
7  print(re.search(pattern2, "123-4567"))  # OK
```

re.findall() 예제

특수 시퀀스

- `\d` : 숫자 하나와 매치 (digit)
- `\w` : 문자, 숫자, 밑줄 (word)
- `\s` : 공백 문자 (space)
- 대문자로 쓰면 반대 의미로 그것들이 아닌 문자를 의미함: (`\D`, `\W`, `\S`)

```
import re
text = "전화: 010-1234-5678"
nums = re.findall(r"\d{3}-\d{4}-\d{4}", text)
print(nums)
```

re.sub()로 치환

```
text = "나쁜말, 나빠!"
clean = re.sub(r"(나쁜말|나빠)", "***", text)
print(clean)
```

이메일과 날짜 추출

```
text = "user@mail.com, 2025-07-05"
emails = re.findall(r"[\w.-]+@[\w.-]+", text)
dates = re.findall(r"\d{4}-\d{2}-\d{2}", text)
```

마무리 및 과제

- 문자열 함수 복습 퀴즈
- 정규표현식으로 텍스트 추출 실습
- 뉴스/댓글/메일에서 정보 추출 실습: (html 태그 지우기)