

빅데이터 수집시스템 개발



문자열 처리와 그룹연산

학습목표

- 문자열로부터 원하는 **문자**를 **추출**하고 **수정** 또는 **제거**할 수 있다.
- **문자열**을 **분해**하여 리스트로 만들거나, 리스트 형태의 **문자열**을 **결합**하여 한 문장을 만들 수 있다.

학습내용

- 문자열 처리
- 그룹연산과 문자열 함수

문자열 처리



1 문자열의 의미

1 문자열 기본

문자열(String)

- 문자, 단어 등으로 구성된 문자들의 집합

- 파이썬에서 문자열은 "(큰따옴표)" 또는 "(작은따옴표)"로 둘러싸아 만듦
- 우리 눈에 숫자로 보이더라도 따옴표 안에 있는 데이터는 문자열임

문자열의 예시

- "대한민국은 민주공화국이다"
- "1234 "
- '도토리 3개'
- 'beautiful day'

문자열 처리



1 문자열의 의미

1 문자열 기본

- 문자열을 여러 줄에 걸쳐 써야 할 경우 큰따옴표를 3개 쓰거나 작은따옴표를 3개 쓰면 됨

```
mystr = ""
```

```
    문장이 한 줄을 초과할 경우에는  
    이런 식으로 작성하면 됨  
""
```

```
""
```

```
좌변에 변수가 없을 때는 문자열이 아니라 여러 줄에 걸친  
주석으로 판단함
```

```
""
```

문자열 처리

1 문자열의 의미

1 문자열 기본

#파일명 : exam12_1.py

#문자열(String)

```
msg1 = "Hello Python"
msg2 = 'Python is easy'
```

```
msg3 = """
나보기가 역겨워 가실 때에는
말없이 고이 보내드리우리다
영변에 약산 진달래꽃 아릅따다
가실길에 뿌리오리다
"""
```

```
print(msg1)
print(msg2)
print(msg3)
```

변수=""" 형태를 유지해야 함
다음 데이터를 바로 써도 되고
한 줄 내려서 기술해도 됨

```
Hello Python
Python is easy
```

```
나보기가 역겨워 가실 때에는
말없이 고이 보내드리우리다
영변에 약산 진달래꽃 아릅따다
가실길에 뿌리오리다
```

문자열 처리

1 문자열의 의미

2 숫자로 구성된 문자열 처리

input 함수

- 데이터를 받아들이면 모두 문자열로 인식

문자열 > 숫자 변환하여 계산

String ➡ int 타입으로 전환

타입 전환

변수 = (전환 타입)수식 형태

```
value1 = int( input("정수를 입력하세요") )
```

```
value2 = float( input("실수를 입력하세요") )
```

만일 입력자가 정수를
입력하지 않으면
예외 발생

문자열 처리

1 문자열의 의미

3 입력된 문자열이 숫자로 이루어졌는지 확인하는 함수

- input 함수를 이용하여 받아들이는 데이터에 숫자가 아닌 문자가 포함되어 있다면 파이썬은 예외를 발생시킴
- 예외를 발생시키지 않고 사전에 오류를 점검하고 싶다면 먼저 **타입 전환이 가능한지 확인**

isdecimal

- 문자열이 숫자로만 구성되었으면 True, 아니면 False 반환

isdigit

- 특수 기호 중 어깨 위에 제곱이나 세제곱을 표시하는 문자처럼 '숫자처럼 생긴' 모든 글자를 다 숫자로 판단

isnumeric()

- 숫자 값 표현에 해당하는 텍스트까지 숫자로 인정
- 예) "½"

문자열 처리

1 문자열의 의미

4 예제

#파일명 : exam12_2.py

```
value1 = input("정수를 입력하세요 ")  
value2 = input("정수를 입력하세요 ")
```

```
#print(value1.isdigit() )  
if value1.isdecimal() and value2.isdecimal():  
    result = value1 + value2  
    print("결과는 ", result, " 입니다 ")
```

- 입력받은 데이터가 숫자로 전환 가능한지 확인
- 둘 다 True일 때 수행, 아니면 정수가 입력되지 않아 연산에 실패했다는 오류 메시지 발생

#위의 코드 수정하기

```
result2 = int(value1) + int(value2)  
print("결과는 ", result2, " 입니다 ")  
else:  
    print("정수를 입력하지 않아서 연산에 실패했습니다")
```


문자열 처리



1 문자열의 의미

4 예제

```
정수를 입력하세요 12
정수를 입력하세요 34
결과는 1234 입니다
결과는 46 입니다
```

- 두 개의 정수를 입력받아 결과를 출력하려고 할 때 12와 34를 입력
 ➔ 결과값이 46이 나오는 것이 아니라 1234가 나옴

문자열 처리



1 문자열의 의미

5 문자열 출력

- 문자열을 출력 시 줄바꿈 또는 특별한 문자 등을 출력하기 위해 **이스케이프(Escape) 문자** 사용

이스케이프(Escape) 문자

- 프로그래밍할 때 사용할 수 있도록 미리 정의해 둔 **문자 조합**
- 출력을 보기 좋게 정렬하는 용도로 사용
- 문자열 안에 함께 사용
- 파이썬에서는 키보드의 우측 상단 백스페이스 아래에 있는 문자 ₩(역슬래쉬)와 함께 사용

문자열 처리



1 문자열의 의미

5 문자열 출력

- 영문 폰트에서는 **\(역슬래쉬)**로 보이고 한글 폰트에서는 **₩(원화)**로 보임

문자	의미
\n	문자열에서 줄바꿈을 할 때 사용
\t	문자열에서 탭 키를 적용할 때 사용
\\	역슬래쉬 문자 자체를 출력하고자 할 때 사용
\'	작은따옴표를 출력하고자 할 때 사용
\"	큰따옴표를 출력하고자 할 때 사용

문자열 처리

1 문자열의 의미

5 문자열 출력

```
print("₩"파이썬₩"은 데이터분석에 사용되는 언어입니다")
print("₩'파이썬₩'은 ₩'딥러닝₩'에도 사용됩니다")
print("파이썬₩n데이터분석₩n크롤링")
print("파이썬₩t데이터분석₩t크롤링")
print("c:₩₩pythonwork_space₩₩exam12₩₩exam12_1.py")
```

```
msg = "파이썬₩t뷰티플스프₩n사이킷런"
print(msg)
```

```
path = r"c:₩pythonwork_space₩exam12₩exam12_1.py"
print(path)
```

문자열 앞에 r을 붙이면 escape 문자가 무력화되어 역슬래시를 두 개씩 붙이지 않아도 됨

- 이스케이프(Escape) 문자를 사용해서 다양한 기능을 구현할 수 있음

```
"파이썬"은 데이터분석에 사용되는 언어입니다
'파이썬'은 '딥러닝'에도 사용됩니다
파이썬
데이터분석
크롤링
파이썬  데이터분석      크롤링
c:\pythonwork_space\exam12\exam12_1.py
파이썬  뷰티플스프
사이킷런
c:\pythonwork_space\exam12\exam12_1.py
```

문자열 처리

1 문자열의 의미

6 문자열 연산

- 문자열을 더하거나 곱할 수 있음
➡ 이 두 가지 연산만 허용됨

더하기 연산

- 두 개의 문자열을 합쳐서 하나의 문자열을 만들

곱하기 연산

- 특정 문자열을 지정한 횟수만큼 반복하여 긴 문자열을 만들 때 사용

```
s = 'Hello'
s2 = 'Python'
s3 = s + " " + s2
print(s3)
```

결과 : Hello Python

```
line = " - " * 5
print(line)
```

결과 ----- 의 문자를 * 뒤에 숫자만큼 반복

문자열 처리

1 문자열의 의미

6 문자열 연산

```
#exam12_4.py
```

```
print()
```

```
s1 = "Hello"
```

```
s2 = "Python"
```

```
s = s1 + " " + s2
```

```
print(s)
```

```
s = s1 * 3
```

```
print(s)
```

```
line = "-" * 50
```

```
print(line)
```

```
for i in range(1, 11):
```

```
    print("*" * i)
```

```
print()
```

```
Hello Python
HelloHelloHello
```

```
-----
*
**
***
****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
```

문자열 처리

2 문자열 인덱싱과 슬라이싱

1 문자열 인덱싱

- 문자열의 요소들을 하나씩 접근할 수 있음
- 문자열 변수[위치값]의 형태로 **데이터를 한 글자씩 읽을 수 있음**
- 값을 **읽기 위한 용도**로만 사용되고 쓰기 위한 용도로는 사용되지 않음
- 원래의 글자 범위를 벗어나면 예외가 발생함
- -를 사용할 수도 있음

```
s = "Hello"
print(s[0] )
print(s[1] )
```

```
s[0] = "h"           #허용되지 않음
#첫 글자를 소문자로 바꾸려면 별도의 문자열 객체를 만들어서
#복사해야 함
```

```
s2 = "h" + s[1] + s[2] + s[3] + s[4]
#여러 개의 글자일 경우 실제로 이렇게 사용하지 않고
#슬라이싱을 사용
```

슬라이싱 예) s2 = "h" + s[1:]

문자열 처리



2 문자열 인덱싱과 슬라이싱

2 문자열 슬라이싱

슬라이싱

- 창문을 열었다 닫았다 하는 것처럼 보여서 붙여지게 됨

인덱싱

한 번에 하나의 글자만 읽을 수 있음

슬라이싱

어디부터 어디까지라고 지정할 수 있음

문자열 처리

2 문자열 인덱싱과 슬라이싱

2 문자열 슬라이싱

- 문자열 변수[시작:종료] 형태로 콜론(:) 연산자를 이용해서 시작과 끝을 지정
- 시작과 콜론만 지정 : 시작 위치부터 마지막 문자까지 문자열 추출 가능
- 콜론과 마지막 위치 지정 : 처음부터 지정된 마지막 인덱스 -1의 위치까지 문자열 추출

```
s = "Hello"
print(s[0:5] )      Hello
print(s[3:] )       Hel , 0부터 시작해서 마지막에 지정한
                    3번을 제외한 나머지 추출
print(s[:3] )       lo , 3번 방부터 시작해서 마지막까지

s = "blue diamond"
s2 = s[0:4]          blue
s3 = s[5:]            diamond
```

문자열 처리

2 문자열 인덱싱과 슬라이싱

3 예제 1

```
#exam12_5.py
str = "Hello Python"
```

```
print(str[0])
print(str[1])
print(str[2])
print(str[3])
print(str[4])
```

```
print(str[-1])
print(str[-2])
print(str[-3])
```

인덱싱은 -를 사용하면 뒤쪽에서부터 시작함

```
print(str[0:5]) #0~5번째 문자열
print(str[:5])
print(str[6:])
```

#인덱싱을 이용한 문자열 뒤집기

```
for i in range(-1, -(len(str)+1), -1):
    print(str[i], end='')
print()
```

```
H
e
l
l
o
n
o
h
Hello
Hello
Python
nohtyP olleH
```

문자열 처리

2 문자열 인덱싱과 슬라이싱

4 예제 2

```
#exam12_6.py
print()
str = "korea"
#str[0] = 'K' - 인덱싱으로 값을 바꿀 수 없음
print(str)
```

```
str = 'K' + str[1:]
print(str)
```

```
str = "python programming"
```

```
s1 = str[0:3] + "h" + str[3:5]
s2 = str[6:12] + "m" + str[12:]
print(s1, s2)
print()
```

문자열 인덱싱은
데이터를 읽기 위한 용도로만 사용,
에러 발생

문자열 추출과 +연산을 통해 새로운
문자열 객체가 만들어져 반환됨

```
korea
Korea
python programming
```

문자열 처리



3 문자열 포매팅

1 문자열 포매팅의 정의

- 출력값이 문자, 숫자(정수, 실수 등), 여러 개의 문자열로 구성되었을 경우 정해진 형태로 출력하고자 하는 것
➡ 파이썬은 두 가지 방식 지원

문자열 처리

3 문자열 포매팅

1 문자열 포매팅의 정의

1 방법 1

`s = "가로 %d 세로 %d인 사각형의 면적은 %d입니다" % (width, height, width*height)`

`%d`

- 서식 문자열로, **문자열 중간에 삽입할 데이터가 있을 때** 그 위치에 놓은 값의 형태에 따라 다른 문자를 씀
- **정수를 그 자리에 놓겠다는 의미**
- "문자열" %(튜플) 형태로 값을 전달
 ➔ 문자열에 있는 기호 %d들에 괄호 안의 값이 차례로 전달됨

문자열 처리



3 문자열 포매팅

1 문자열 포매팅의 정의

2 방법 2

```
s = "가로 {} 세로 {}인 사각형의 면적은 {}입니다.".format  
    (width, height, width*height)
```

- {} 위치에 값들이 투입됨

문자열 처리



3 문자열 포매팅

2 문자열 포매팅 형식

코드	설명
%d	정수
%f	실수
%s	문자열
%c	하나의 문자
%o	8진수
%x	16진수
%%	%기호 자체를 출력하고자 할 때

문자열 처리

3 문자열 포매팅

예제 1

#정수는 d

```
s = "결과는 %d 입니다." % 10  
print(s)
```

```
s = "안녕하세요 %s님" % "홍길동"  
print(s)
```

#여러 개

```
s = "%s 님의 주급은 %d 입니다." % ("홍길동", 500000)  
print(s)
```

- 여러 개의 데이터를 문자열 중간에 넣으려면 % 뒤에 데이터를 튜플 형식으로 전달
- 튜플은 여러 개의 데이터를 하나로 묶어주는 타입으로 반드시 괄호를 해야 함

(계속)

문자열 처리

3 문자열 포매팅

3 예제 1

#실수는 f를 사용

```
print("%f X %f = %f" % (2.4, 3.5, 2.4*3.5))
```

```
print("%7.2f X %7.2f = %7.2f" % (2.4, 3.5, 2.4*3.5))
```

```
print("현재 목표 달성율은 %d%% 입니다." % 50)
```

#정렬

```
print("%5d*" % 12)
```

#5자리 확보 후 숫자를 오른쪽부터 채워서 출력

```
print("%-5d*" % 12)
```

#5자리 확보 후 숫자를 왼쪽부터 채워서 출력

```

결과는 10 입니다.
안녕하세요 홍길동님
홍길동 님의 주급은 500000 입니다.
2.400000 X 3.500000 = 8.400000
    2.40 X    3.50 =    8.40
현재 목표달성율은 50% 입니다.
    12*
12    *
```

문자열 처리

3 문자열 포매팅

4 예제 2

```
#exam12_8.py
```

```
print("{} + {} = {}".format(3,4,3+4))
print("{1} + {0} = {2}".format(3,4,3+4))
print("{:.2f} X {:.2f} = {:.2f}".format(2.4, 3.5, 2.4*3.5))
print("{} is {}".format("flower", "beautiful"))

print("{:<5d}*".format(12))
print("{:>5d}*".format(12))

for i in range(1,6):
    print("{:04d}".format(i))
```

" ".format()
함수

- 문자열에 여러 가지 형태의 데이터를 끼워 넣어 하나의 문자열을 만들 때 사용

문자열 처리

3 문자열 포매팅

4 예제 2

"{0} {1} {2}".format(3, 4, 5) 형태로 기술

- {} 기호 안에 값이 들어감
- {} 개수만큼 format 함수에 값을 전달
- {0} {2} {1}으로 숫자 지정
➔ 전달해주는 값의 순서에 맞게 차례대로 전달

자릿수 지정은 { :4d } 형태로 기술

- :뒤에 4d라고 쓰면 정수 들어갈 자리 4자리 확보 후 오른쪽 끝부터 채움
- 왼쪽 정렬을 하려면 {:>4d} 형태로 기술
- {:04d}는 자릿수를 4자리로 맞추고 오른쪽부터 숫자를 채운 후 빈 공간을 0으로 채움

그룹연산과 문자열 함수



1 문자열의 위치 찾기

count 함수 사용

- 특정 문자열이 **몇 번 존재하는지** 찾기 위해 사용
- 찾는 문자열이 몇 번 등장했는지를 반환함

index 함수 사용

- 특정 문자열이 **어디에 존재하는지** 찾기 위해 사용
- 문자열의 위치를 반환
- 찾는 문자열이 없을 경우 : 예외를 반환
 - ➡ count 함수나 in 연산자를 활용해 먼저 데이터가 있는지 확인 후 index 함수 사용

그룹연산과 문자열 함수

1 문자열의 위치 찾기

```
s="I like star, red star yello star, blue star"
```

```
print("star 개수 :", s.count("star"))
print("star 단어 존재 여부 :", "star" in s)
print("star 첫 번째 위치 :", s.index("star"))
print("star 두 번째 위치 :", s.index("star", 12))
```

- index 함수에 존재하지 않는 값을 전달할 때 에러 메시지

```
Traceback (most recent call last):
  File "exam12_9.py", line 54, in <module>
    print( s.index("gray"))
ValueError: substring not found
```

- count 함수와 index 함수를 이용해서 문자열 내의 특정 문자열을 모두 찾아보는 코드 작성

```
s="I like star, red star yello star, blue star"
```

```
print("star 위치")
key = "star"
pos=0
while s.count(key, pos)!=0:
    pos = s.index(key, pos)
    pos = pos + len(key) #단어 길이만큼 다음 위치로 이동함
    print(pos)
```

그룹연산과 문자열 함수

2 문자열의 분리와 결합

split 함수

- 문자열을 **특정 문자를 기준으로 분리**하여 문자열의 list 형태로 반환
- 매개변수에 특정 문자를 지정하지 않으면 공백으로 토큰을 나눔

```
s="I like star red star yellow star blue star"
```

```
words = s.split()  
print(words) 리스트 형태임
```

```
s = "소나무,전나무,오동나무,사시나무,향나무"  
s2 = s.split(",")  
print(s2)
```

그룹연산과 문자열 함수



2 문자열의 분리와 결합

join 함수

- join의 문자열 리스트를 전달받아 하나의 문자열을 만들 때 사용

#문자열 결합

```
s = ",".join(["apple", "orange", "banana", "mango", "cherry"])
print(s)
```

```
s = " ".join(["apple", "orange", "banana", "mango", "cherry"])
print(s)
```

```
apple,orange,banana,mango,cherry
apple orange banana mango cherry
```

그룹연산과 문자열 함수

3 그 밖의 함수들

upper 함수

- 문자열을 **대문자**로 변환하는 함수

lower 함수

- 문자열을 **소문자**로 변환하는 함수

replace 함수

- 문자열을 **특정 문자열**로 전환하는 함수

```
s="I like star, red star yello star, blue star"
```

```
s = s.upper()
```

```
print(s)
```

결과 : I LIKE STAR RED STAR YELLO STAR BLUE STAR

```
s = s.lower()
```

```
print(s)
```

결과 : i like star red star yello star blue star

```
s=s.replace("star", "moon")
```

```
print(s)
```

결과 : i like moon red moon yello moon blue moon

그룹연산과 문자열 함수



3 그 밖의 함수들

- 문자열의 앞·뒤에 있는 공백문자를 제거하기 위해 **strip 함수**를 사용

strip 함수

- 양쪽을 다 제거할 때

rstrip 함수

- 왼쪽의 공백만 제거

lstrip 함수

- 오른쪽 공백만 제거

```
s = "  space remove  "
print("*" + s + "*")
print("*" + s.lstrip() + "*")
print("*" + s.rstrip() + "*")
print("*" + s.strip() + "*")
```

```
*  space remove  *
*space remove  *
*  space remove*
*space remove*
```

문자열 앞뒤로 공백이 있다
 문자열 왼쪽의 공백이 제거되었다
 문자열 오른쪽의 공백이 제거되었다
 문자열 양쪽의 공백이 제거되었다

그룹연산과 문자열 함수



3 그 밖의 함수들



```
#exam12_9.py
s="I like star, red star yellow star, blue star"
words = s.split()
print(words )

s1 = "소나무,전나무,오동나무,사시나무,향나무"
woods = s1.split(",")
print(woods)

print("문자열 길이 : ", len(s) )
```

#특정 문자의 개수

```
print ("star 개수 : ", s.count("star"))
```

#star의 위치

```
print("star 첫 번째 위치 : ", s.index("star"))
```

#모든 star 찾기

```
print("star 위치")
key = "star"
pos=0
while s.count(key, pos)!=0:
    pos = s.index(key, pos)
    pos = pos + len(key) #단어 길이만큼 다음 위치로 이동
    print(pos)
```

(계속)

그룹연산과 문자열 함수



3 그 밖의 함수들



```
print("소문자 대문자 체인지")  
s2 = s.upper()  
print(s2)
```

```
s1 = s2.lower()  
print(s1)
```

#문자열 대체

```
#star를 moon으로 대체  
s = s.replace("star", "moon")  
print(s)
```

#공백 제거

```
s = "  space remove  "  
print("*" + s.lstrip() + "*")  
print("*" + s.rstrip() + "*")  
print("*" + s.strip() + "*")
```

#문자열 삽입

```
s = ",".join(["apple", "orange", "banana", "mango", "cherry"])  
print(s)  
s = " ".join(["apple", "orange", "banana", "mango", "cherry"])  
print(s)
```

그룹연산과 문자열 함수



3 그 밖의 함수들

```
[ 'I', 'like', 'star', 'red', 'star', 'yello', 'star', 'blue', 'star' ]
[ '소나무', '전나무', '오동나무', '사시나무', '향나무' ]
문자열 길이 : 41
star 개수 : 4
star 첫번째 위치 : 7
star 위치
11
20
31
41
소문자 대문자 체인지
I LIKE STAR RED STAR YELLO STAR BLUE STAR
i like star red star yello star blue star
I like moon red moon yello moon blue moon
*   space remove   *
*space remove      *
*   space remove*
*space remove*
apple,orange,banana,mango,cherry
apple orange banana mango cherry
```

학습정리

1. 문자열 처리



- 문자열은 인덱싱과 슬라이싱을 통해 접근하거나 추출함
- 파이썬에서 인덱싱을 이용하거나 슬라이싱을 이용해 데이터를 추출할 수 있지만, 데이터의 수정은 불가능함
- 인덱싱과 슬라이싱한 추출한 문자열과 다른 문자열을 결합하여 새로운 문자열을 만들 수 있음
- index 함수는 서브 문자열의 위치를 알려주는 함수지만, 문자열이 없을 경우 예외 발생
- 프로그램상에서 실제로 적용할 경우 count 함수를 이용해 단어가 존재하는지 확인하거나 in 연산자를 이용해서 확인한 후 사용
- 포매팅(%나 format 함수)으로 다양한 데이터들을 하나의 문자열로 만들어 출력을 아름답게 꾸밀 수 있음
- 가급적 format 함수를 사용하는 것이 좋음

학습정리

2. 그룹연산과 문자열 함수



- 문자열이 숫자만으로 구성되었는지 확인하는 함수에는 isdigit, isdecimal, isnumeric 함수가 있음
- 문자열 양쪽의 쓸데없는 공백을 제거하기 위해서는 strip(양쪽), lstrip(왼쪽),rstrip(오른쪽) 함수를 사용함