

데이터 과학을 위한

파이썬  
프로그래밍



## 06. 문자열

# 목차

1. 문자열의 이해
2. Lab: 단어 카운팅
3. 문자열 서식 지정

01

# 문자열의 이해

# 01. 문자열의 이해

## ■ 문자열의 개념

- 문자열은 시퀀스 자료형(sequence data type)이다.

```
a = "abcde"
```

a	0100 1001
b	0100 1010
c	0100 1011
d	0100 1100
e	0100 1101

[시퀀스 자료형]

# 01. 문자열의 이해

## ■ 문자열과 메모리 공간

- 일반적으로 문자열을 저장하기 위해서는 영문자 한 글자당 1바이트의 메모리 공간을 사용한다.
- 다음과 같은 코드로 문자열이 저장된 공간의 크기를 확인할 수 있다.

```
>>> import sys          # sys 모듈을 호출
>>> print(sys.getsizeof("a"), sys.getsizeof("ab"), sys.getsizeof("abc"))
50 51 52                # "a", "ab", "abc" 각각의 메모리 크기 출력
```

- 컴퓨터에 a라고 알려 줘도 컴퓨터는 정확히 a라는 텍스트를 인식하는 것이 아니다. 대신 컴퓨터는 이 정보를 이진수로 변환하여 저장한다.

# 01. 문자열의 이해

## ■ 문자열과 메모리 공간

- 컴퓨터 공학자들은 이러한 문자를 처리하기 위해 이진수로 변환되는 표준 규칙을 만들었다. ASCII, CP949, MS949, UTF-8 등 이러한 규칙을 인코딩(encoding)이라고 한다.
  - ① 컴퓨터는 문자를 직접 인식하지 못한다.
  - ② 컴퓨터는 문자를 숫자로 변환하여 인식한다.
  - ③ 사람들은 문자를 숫자로 변환하기 위한 규칙을 만들었다.
  - ④ 일반적으로 이 규칙은 1개의 영문자를 1바이트, 즉 2의 8승(28) 정도의 공간에 저장될 수 있도록 정하였다.

# 변수의 메모리 공간 사용- 간단한 실험

```
In[2]: import sys
In[3]: a='a'
In[4]: b='ab'
In[5]: sys.getsizeof(a)
Out[5]: 50
In[6]: sys.getsizeof(b)
Out[6]: 51
In[7]: sys.getsizeof('a')
Out[7]: 50
In[8]: sys.getsizeof('12345')
Out[8]: 54
```

- `sys.getsizeof(변수)`
  - 변수를 저장하는데 필요한 메모리 용량(byte)를 반환하는 함수
- 현재 영문 스트링 변수 1개를 저장하는데 50바이트, 한글 스트링변수는 76바이트가 소요되고 있다.
- 빈 문자열에 대해서는 51바이트

```
In[3]: a = '한'
In[4]: b = '한글'
In[5]: sys.getsizeof(a)
Out[5]: 76
In[6]: sys.getsizeof(b)
Out[6]: 78
In[7]: c = ''
In[8]: sys.getsizeof(c)
Out[8]: 51
In[9]: d = 'a'
In[10]: sys.getsizeof(d)
Out[10]: 50
In[11]: e = 1
In[12]: sys.getsizeof(e)
Out[12]: 28
In[13]: f = 256
In[14]: sys.getsizeof(f)
Out[14]: 28
In[15]: g = 65536
In[16]: sys.getsizeof(g)
Out[16]: 28
```

# 01. 문자열의 이해

## ■ 문자열과 메모리 공간

000	(nul)	016	► (dle)	032	sp	048	0	064	0	080	P	096	`	112	p
001	Ⓢ (soh)	017	◄ (dcl)	033	!	049	1	065	A	081	Q	097	a	113	q
002	Ⓢ (stx)	018	‡ (dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003	♥ (etx)	019	‡ (dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004	♦ (eot)	020	⌘ (dc4)	036	\$	052	4	068	D	084	T	100	d	116	t
005	♣ (enq)	021	§ (nak)	037	%	053	5	069	E	085	U	101	e	117	u
006	♣ (ack)	022	— (syn)	038	&	054	6	070	F	086	V	102	f	118	v
007	• (bel)	023	‡ (etb)	039	'	055	7	071	G	087	W	103	g	119	w
008	■ (bs)	024	↑ (can)	040	(	056	8	072	H	088	X	104	h	120	x
009	(tab)	025	↓ (em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011	Ⓢ (vt)	027	← (esc)	043	+	059	;	075	K	091	[	107	k	123	{
012	Ⓢ (np)	028	L (fs)	044	,	060	<	076	L	092	\	108	l	124	
013	(cr)	029	↔ (gs)	045	-	061	=	077	M	093	]	109	m	125	}
014	♣ (so)	030	▲ (rs)	046	.	062	>	078	N	094	^	110	n	126	~
015	Ⓢ (si)	031	▼ (us)	047	/	063	?	079	O	095	_	111	o	127	o

[UTF-8의 유니코드(출처: Nicolas Bouliane)]

### UTF-8: Universal Coded Character Set + Transformation Format – 8-bit

유니코드를 위한 가변 길이 문자 인코딩 방식 중 하나이다.

유니코드 한 문자를 나타내기 위해 1바이트에서 4바이트까지를 사용한다.

예를 들어서, U+0000부터 U+007F 범위에 있는 [ASCII](#) 문자들은 UTF-8에서 1바이트만으로 표시된다.



# 01. 문자열의 이해

## ■ 문자열의 인덱싱

- 리스트처럼 글자 하나하나가 상대적인 주소(offset)를 가지는데, 이 주소를 사용해 할당된 값을 가져오는 인덱싱을 사용할 수 있다.

Hello

0	1	2	3	4
-5	-4	-3	-2	-1

인덱스 번호

[문자열의 처리]

```
>>> a = "abcde"
>>> print(a[0], a[4])           # a 변수의 0번째, 4번째 주소에 있는 값
a e
>>> print(a[-1], a[-5])        # a 변수의 오른쪽에서 0번째, 4번째 주소에 있는 값
e a
```

# 01. 문자열의 이해

## ■ 문자열의 슬라이싱

- 슬라이싱(slicing) : 문자열의 주소값을 기반으로 문자열의 부분 값을 반환하는 기법이다.

```
>>> a = "TEAMLAB MOOC, AWESOME Python"
>>> print(a[0:6], " AND ", a[-9:])          # a 변수의 0부터 5까지, -9부터 끝까지
TEAMLA AND ME Pyhon
>>> print(a[:])                             # a 변수의 처음부터 끝까지
TEAMLAB MOOC, AWESOME Python
>>> print(a[-50:50])                        ✓ # 범위를 넘어갈 경우 자동으로 최대 범위를 지정
TEAMLAB MOOC, AWESOME Python
>>> print(a[::2], " AND ", a[::-1])
TALBMO,AEOEpto AND nohtyP EMOSEWA ,COOM BALMAET
```

시작\_인덱스:끝\_인덱스번호+1

시작\_인덱스:끝\_인덱스번호+1:증분

증분=-1이면 역순으로 접근

지정하지 않으면 맨 앞

지정하지 않으면 맨 뒤

지정하지 않으면 +1

# 01. 문자열의 이해

## ■ 문자열의 연산

- 가장 기본적인 연산은 리스트의 연산과 같다. 예를 들어, 문자열 변수 'a'와 정수형인 2의 'a+2'와 같은 연산은 동작하지 않는다. 하지만 'a\*2'와 같은 연산은 지원한다.

```
>>> a = "TEAM"
>>> b = "LAB"
>>> print(a + " " + b)           # 덧셈으로 a와 b 변수 연결하기
TEAM LAB
>>> print(a * 2 + " " + b * 2)    # 곱하기로 반복 연산 가능
TEAMTEAM LABLAB
>>> if 'A' in a: print(a)          # 'A'가 a에 포함되었는지 확인
...     else: print(b)
...
TEAM
```

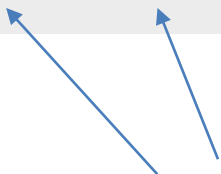
# 01. 문자열의 이해

## ■ 문자열의 연산

- 다음과 같이 코드를 작성하면 문자열과 정수형의 연산으로 인식하여 덧셈 연산이 실행되지 않는다. 연산이 가능한 조합의 데이터형에 대해서 연산이 가능하다.

```
>>> int_value = 2  
>>> print("결과는" + int_value)
```

string + int 연산으로 오류 발생



참고: 실행 가능한 코딩 사례

```
print("결과는", int_value)  
print("결과는" + str(int_value))  
print(f"결과는 {int_value}")
```

# 01. 문자열의 이해

## ■ 문자열에 지원되는 함수 methods

함수명	기능
len()	문자열의 문자 개수를 반환
upper()	대문자로 변환
lower()	소문자로 변환
title()	각 단어의 앞글자만 대문자로 변환
capitalize()	첫 문자를 대문자로 변환
count('찾을 문자열')	'찾을 문자열'이 몇 개 들어 있는지 개수 반환
find('찾을 문자열')	'찾을 문자열'이 왼쪽 끝부터 시작하여 몇 번째에 있는지 반환
rfind('찾을 문자열')	find() 함수와 반대로 '찾을 문자열'이 오른쪽 끝부터 시작하여 몇 번째에 있는지 반환
startswith('찾을 문자열')	'찾을 문자열'로 시작하는지 여부 반환
endswith('찾을 문자열')	'찾을 문자열'로 끝나는지 여부 반환

# 01. 문자열의 이해

## ■ 문자열에 지원되는 함수 methods

함수명	기능
strip()	좌우 공백 삭제
rstrip()	오른쪽 공백 삭제
lstrip()	왼쪽 공백 삭제
split()	문자열을 공백이나 다른 문자로 나누어 리스트로 반환
isdigit()	문자열이 숫자인지 여부 반환
islower()	문자열이 소문자인지 여부 반환
isupper()	문자열이 대문자인지 여부 반환

참고:

파이썬 스트링 메소드 [https://www.w3schools.com/python/python\\_ref\\_string.asp](https://www.w3schools.com/python/python_ref_string.asp)

# String method 연습

@str\_methods.py

```
11 a = 'Hello world'
12 print("len(a)=", len(a))           # This is a intrinsic python function.
13
14 print("1) 문자열의 대소 문자 바꾸기 및 문자열 출현 회수 세기")
15 print("a.upper()=", a.upper())     # a.upper()= HELLO WORLD
16 print("a.lower()=", a.lower())     # a.lower()= hello world
17 print("a.title()=", a.title())     # a.title()= Hello World. # 각 단어의 앞글자만 대문자로 변환
18
19 b = 'abc bcd efg abc'.capitalize() # 스트링의 첫 글자만 대문자로 변환. Abc bcd efg abc
20                                     ↑   ↑   ↑   ↑
21 # 문자의 출현 횟수 반환
22 print("b.count('abc')=", b.count('abc')) # b.count('abc')= 1. 대소문자 가림
23 print("b.count('bc')=", b.count('bc'))   # b.count('bc')= 3. ↑ 스트링의 중간에 있어도 반영함.
24 str = "012345012345...." # 문자열을 검색하여 발견한 회수를 반환한다.
25 print(str.count('.'), str.count('0'), str.count('5.')) # 4 2 1
```

len(a)= 11

1) 문자열의 대소 문자 바꾸기 및 문자열 출현 세기

a.upper()= HELLO WORLD

a.lower()= hello world

a.title()= Hello World

b.count('abc')= 1

b.count('bc')= 3

4 2 1

# String method 연습

@str\_methods.py

```
28 print("\n2) 문자열 찾아 위치 반환하기")
29 b = 'abc bcd efg abc'
30
31 # find(str, start, end): 부분 문자열 str의 최초 위치 (첫 글자의 인덱스)를 반환. 왼쪽부터 검색.
32 #   찾을 문자열이 몇 번째 있는지 반환. 0번부터 시작. 처음 만난 문자열의 위치만 반환한다.
33 #   못 찾으면 -1을 반환한다. 유사한 함수인 index()는 못 찾으면 오류 발생하여 종료한다.
34 #   start와 end에는 검색을 원하는 첫 인덱스, 마지막 인덱스를 지정.
35 #   지정하지 않을 경우, start의 default 값은 0, end의 default의 값은 제일 마지막 인덱스.
36
37 print("b.find('abc')=", b.find('abc'))      # b.find('abc')= 0
38 print("b.find('cd')=", b.find('cd'))        # b.find('cd')= 5
39 print("b.find('bc')=", b.find('bc'))        # b.find('bc')= 1
40 print("b.find('g')=", b.find('g'))          # b.find('g')= 10
41 print("b.find('abcd')=", b.find('abcd'))    # b.find('abcd')= -1. 없음.
42
43 # index(str, beg=0, end=len(string))
44 #   Same as find(), but raises an exception if str not found.
45 print("b.index('cd')=", b.index('cd'))      # 추가: b.index('cd')= 5
46 # print("b.index('abcd')=", b.index('abcd')) # 발견되지 못해 exception 발생. 수행중지
47
48 # rindex( str, beg=0, end=len(string)) 뒤에서부터 찾되 index 번호는 앞에서부터 센 것으로 반환한다.
49 #   Same as index(), but search backwards in string.
50 print("b.rindex('a')=", b.rindex('a'))      # 추가: b.index('a')= 12
51 print("b.rindex('efg')=", b.rindex('efg'))  # 추가: b.index('efg')= 8
```

2) 문자열 찾아 위치 반환하기  
b.find('abc')= 0  
b.find('cd')= 5  
b.find('bc')= 1  
b.find('g')= 10  
b.find('abcd')= -1  
b.index('cd')= 5  
b.rindex('a')= 12  
b.rindex('efg')= 8



# String method 연습

@str\_methods.py

```
54 print("\n3) 특정 문자로 시작하거나 특정 문자로 끝나는지의 여부 판단하기")
55
56 # 특정 문자열로 시작하는가?
57 print("'Hey'.startswith('He')=", 'Hey'.startswith('He')) # 'Hey'.startswith('He')= True
58 print("'No'.startswith('n')=", 'No'.startswith('n')) # 'No'.startswith('n')= False. 대소문자 구분
59 print("'Hey'.endswith('y')=", 'Hey'.endswith('y')) # 'Hey'.endswith('y')= True
60 print("'Hey'.endswith('ey')=", 'Hey'.endswith('ey')) # 'Hey'.endswith('ey')= True
61 print("'No'.endswith('n')=", 'No'.endswith('n')) # 'No'.endswith('n')= False
62
63 print("\n4) 문자열의 정체 확인: 숫자, 대소문자")
64
65 # isdigit(): Returns true if string contains only digits and false otherwise.
66 print('12345'.isdigit()) # True. 숫자(양의 정수) 이면 True
67 print('-12345'.isdigit(), '1.2'.isdigit()) # False False 음수, 부동소수는 안됨
68 print('abcdE'.islower()) # False. E 때문에 안됨
69 print('123'.isalpha(), 'abc'.isalpha()) # False True
70 print('+12'.isdecimal(), '123'.isdecimal()) # False True
71 # isnumeric(): Returns true if a unicode string contains only
72 # numeric characters and false otherwise.
73 print('1.2'.isnumeric(), '-12'.isnumeric()) # False False
```

3) 특정 문자로 시작하거나 특정 문자로 끝나는지의 여부 판단하기

```
'Hey'.startswith('He')= True
'No'.startswith('n')= False
'Hey'.endswith('y')= True
'Hey'.endswith('ey')= True
'No'.endswith('n')= False
```

4) 문자열의 정체 확인: 숫자, 대소문자

```
True
False False
False
False True
False True
False False
```

# String method 연습

@str\_methods.py

```
76 print("\n5) 스트링의 공백 제거")
77 c = '      Hi      '      # 좌측에 공백문자 5개, 우측에 2개.
78 # lstrip(): Removes all leading whitespace in string.
79 # rstrip(): Removes all trailing whitespace of string.
80 # strip([chars]): Performs both lstrip() and rstrip() on string.
81 print('###' + c.strip() + '???')      # 좌우측의 공백 문자를 제거 ###Hi???
82 print('###' + c.lstrip() + '???')      # 좌측 공백 문자 제거 ###Hi   ???
83 print('---' + c.rstrip() + '???')      # 우측 공백 문자 제거 ---      Hi???
84
85 print("\n6) 문자열 특정 부분을 변경")
86 # replace(old, new [, max]): Replaces all occurrences of old in string
87 # with new or at most max occurrences if max given.
88 print('aa\n\na a\na a\na'.replace('\n', ''))      # aaa aa aa 모두 바꾼다.
89 print('abc aabdab abc'.replace('ab', 'Z', 3))      # Zc aZdZ abc 최대 3개 바꾼다.
90
91 print("\n7) 스트링을 왼쪽 혹은 오른쪽으로 정렬하기")
92 # ljust(width[, fillchar]): Returns a space-padded string with the original string
93 # left-justified to a total of width columns.
94 str = "string example...."
95 print(str.ljust(30, '0'))      # string example....0000000000000
96 print(str.rjust(30))           # string example....
```

5) 스트링의 공백 제거

###Hi???

###Hi ???

--- Hi???

6) 문자열 특정 부분을 변경

aaa aa aa

Zc aZdZ abc

7) 스트링을 왼쪽 혹은 오른쪽으로 정렬하기

string example....0000000000000

string example....

# String method 연습

@str\_methods.py

```
99  print("\n8) 문자열을 단어별로 나누거나 모으기")
100
101  # 공백을 기준으로 문자열을 분리하여 리스트로 반환
102  print('I am a boy.'.split())    # ['I', 'am', 'a', 'boy.']
103
104  time_stamp = "12:47:08"
105  a = time_stamp.split(':')        # :를 기준으로 문자열을 분리하여 list로 반환
106  print(a, type(a))               # ['12', '47', '08'] <class 'list'>
107  b = ':'.join(a)                 # :를 기준으로 리스트 자료를 문자열로 생성
108  print(b, type(b))               # 12:47:08 <class 'str'>
109
110  a = time_stamp.partition(':')
111  print(a, type(a)) # ('12', ':', '47:08') <class 'tuple'> 튜플로 반환한다.
112
113  # 참고: 문자열을 개별 문자로 나누기
114  a = list('I love U.')           # 스트링에 list() 함수를 씌우면 list를 생성함.
115  print(type(a), a) # <class 'list'> ['I', ' ', 'l', 'o', 'v', 'e', ' ', 'U', '.']
116
117  b = ''.join(a)                  # 리스트 내의 원소를 모아서 스트링으로 만들기. ''
118  print(b, type(b)) # I love U. <class 'str'>
```

```
8) 문자열을 단어별로 나누거나 모으기
['I', 'am', 'a', 'boy.']
['12', '47', '08'] <class 'list'>
12:47:08 <class 'str'>
('12', ':', '47:08') <class 'tuple'>
<class 'list'> ['I', ' ', 'l', 'o', 'v', 'e', ' ', 'U', '.']
I love U. <class 'str'>
```

# 01. 문자열의 이해

설명 생략

## ■ 문자열의 연산

- **upper( ) 함수** : 문자열을 대문자로 변환하는 함수
- **lower( ) 함수** : 소문자로 변환하는 함수
- 참고로 문자열 함수를 사용하는 방법은 문자열 변수 다음에 '.문자열 함수'를 입력하면 된다.

```
>>> title = "TEAMLAB X Inflearn"
>>> title.upper()           # title 변수를 모두 대문자로 변환
'TEAMLAB X INFLEARN'
>>> title.lower()          # title 변수를 모두 소문자로 변환
'teamlab x inflearn'
```

# 01. 문자열의 이해

설명 생략

## ■ 문자열의 연산

- **title( ) 함수** : 영어신문의 헤드라인처럼 각 단어의 앞글자만 대문자로 바꾸는 함수
- **capitalize( ) 함수** : 첫 번째 글자만 대문자로 바꾸는 함수

```
>>> title = "TEAMLAB X Inflearn"
>>> title.title()                # title 변수의 각 단어의 앞글자만 대문자로 변환
'Teamlab X Inflearn'
>>> title.capitalize()          # title 변수의 첫 번째 글자만 대문자로 변환
'Teamlab x inflearn'
```

# 01. 문자열의 이해

설명 생략

## ■ 문자열의 연산

- **count( ) 함수** : 해당 문자열에서 특정 문자가 포함된 개수를 반환
- **isdigit( ) 함수** : 해당 문자열이 숫자인지를 True 또는 False로 값을 반환
- **startswith( ) 함수** : 해당 문자열로 시작하는지를 True 또는 False로 값을 반환

```
>>> title = "TEAMLAB X Inflearn"
>>> title.count("a")           # title 변수에 'a'가 몇 개 있는지 개수 반환
1
>>> title.upper().count("a")   # title 변수를 대문자로 만든 후, 'a'가 몇 개 있는지 개수 반환
0
>>> title.isdigit()           # title 변수의 문자열이 숫자인지 여부 반환
False
>>> title.startswith("a")     # title 변수가 'a'로 시작하는지 여부 반환
False
```

# 01. 문자열의 이해

## 여기서 잠깐! 문자열 표현과 특수문자

- 파이썬에서 문자열을 표현할 때 작은따옴표나 큰따옴표를 사용한다. 하지만 다음과 같이 아포스트로피(')가 문장에 들어가면 작은따옴표를 사용하기 어렵다. 만약, 작은따옴표로 문자열을 표현한다면 인터프리터는 이 문자가 제대로 닫히지 않았다고 판단하고 오류를 출력할 것이다

```
It's OK.
```

- 이러한 문제를 지원하기 위해 파이썬에서는 다양한 기능을 제공한다. 먼저 문자열 자체에 작은따옴표나 큰따옴표가 들어가 있는 경우이다.




```
a = "It's OK."
```

# 01. 문자열의 이해

[필독: Python 3 Escape Sequences](#)

## 여기서 잠깐! 문자열 표현과 특수문자

- 다음으로 파이썬의 escape sequence 특수문자 기능을 사용하는 것이다. 아래의 특수문자를 사용할 경우 다음과 같이 아포스트로피(')를 사용할 수 있다.

특수문자	기능	특수문자	기능
\ 	다음 줄과 연속임을 표현	\b	백스페이스
\\	문자 자체	\n	줄 바꾸기
\'	'문자	\t	 키
\"	"문자	\e	 키

```
a = "It's OK."
```

```
In[6]: a="I'm OK"
In[7]: b='I\'m OK'
In[8]: a
Out[8]: "I'm OK"
In[9]: b
Out[9]: "I'm OK"
```



## 참고: Python 3 Escape Sequences

@escape\_sequence.py

```
5 # Backslash and newline ignored. \ 이후에는 주석문 불가. space도 안됨.
6 print("1. line1 \
7   line2 \
8   line3")      # 1. line1 line2 line3
9
10 # literal character. 문자 그대로 하라..
11 print("2. \\")    # 2. \      \ 이후에는 escape sequence가 아니라 문자 \ 이다.
12 print('3. \'')    # 3. '      \ 이후에는 따옴표 '가 아니라 문자 ' 이다.
13 print('4. \\'')   # 4. "      \ 이후에는 따옴표 "가 아니라 문자 " 이다.
14
15 # control character. 장치, 커저 제어 용도...
16
17 # \r: carriage return(CR)   커저를 그 줄 맨 앞으로 보내기.
18 # 2345는 남아있어야 하는데 파이썬에는 모두 지워진다.
19 print("5. Hello12345\rWorld!")  # World!
20
21 # \n: linefeed(LF)   줄 바꾸고, 맨 앞으로 커저를 보낸다.
22 print("6. Hello\nWorld!")      # 6. Hello
23                                # World!
24 # \b: backspace
25 print("7. Hello\bWorld!")      # 7. HellWorld!
26
27 # /t: tab
28 print("8. Hello\tWorld!")      # 8. Hello  World!
```

# 01. 문자열의 이해

## 여기서 잠깐! 문자열 표현과 특수문자

- 또 다른 문제로는 다음과 같은 줄 바꿈 표현이 있다. 이러한 경우에도 문자열로 표현하기 어렵다.

```
It's Ok.  
I'm Happy.  
See you.
```

- 두 줄 이상의 표현도 마찬가지로 표현할 수 있다. 하나는 큰따옴표(")나 작은따옴표(')를 3개로 연결하는 방법이다. 다음과 같이 선언한다

```
a = """"  
It's Ok.  
I'm Happy.  
See you.""""
```

02

Lab: 단어 카운팅

## 02. Lab: 단어 카운팅

yesterday.py + yesterday.txt

### ■ 실습 내용

- 앞에서 배운 문자열의 여러 기능을 사용하여 단어 카운팅 프로그램을 만들어 보자.
- 이번에 진행할 Lab은 팝 그룹 비틀스의 <Yesterday>라는 노래에서 'Yesterday'라는 단어가 몇 번 나오는지 맞추는 단어 카운팅 프로그램이다.

```
f = open("yesterday.txt", 'r')
yesterday_lyric = f.readlines()
f.close()
```

### ■ 실행 결과



```
Number of a Word 'Yesterday' 9
```

## 02. Lab: 단어 카운팅

yesterday.py + yesterday.txt

### ■ 문제 해결

파일 읽고 쓰기: <https://wikidocs.net/26>

파일 - 텍스트 파일 다루기: <https://wikidocs.net/16077>

코드 6-1 yesterday.py

```
1 f = open("yesterday.txt", 'r')
2 yesterday_lyric = f.readlines()
3 f.close()
4
5 contents = ""
6 for line in yesterday_lyric:
7     contents = contents + line.strip() + "\n"
8
9 n_of_yesterday = contents.upper().count("YESTERDAY")
10 print("Number of a Word 'Yesterday'" , n_of_yesterday)
```

## 02. Lab: 단어 카운팅

### ■ 문제 해결 : [코드 6-1] 해석

- 1~3행 : 'yesterday.txt' 파일에서 모든 내용을 불러와 yesterday\_lyric 리스트로 저장한다.
- 5~7행 : for문을 사용하여 yesterday\_lyric 리스트의 내용을 한 줄씩 불러오면서 contents 변수에 저장한다. 그러면 contents 변수에는 <yesterday> 노래의 모든 가사가 저장된다.
- 9~10행 : contents 변수에 있는 값은 모두 대문자로 바꾸는 upper( ) 함수를 사용하여 대문자로 변환한 후, count( ) 함수를 사용하여 대문자 'YESTERDAY'가 몇 개인지를 확인한다. 여기서 contents.upper( ).count("YESTERDAY") 함수를 붙여 써도 작동하는 이유는 upper( ) 함수의 경우 contents 변수에 값 자체를 변경하는 것이 아니라, 변경된 값을 반환해 주는 함수일 뿐이기 때문이다. 다음 파이썬 셸의 코드를 보면 이해하기 쉽다

```
>>> title = "teamlab"
>>> title
'teamlab'
>>> title.upper()
'TEAMLAB'
```

```
>>> title
'teamlab'
```

03

문자열 서식 지정

## 03. 문자열 서식 지정

### ■ 서식 지정의 개념

- `print( )` 함수를 사용하다 보면 어떤 형식에 맞추어 결과를 출력해야 할 일이 발생하기도 한다. 특히 엑셀을 사용할 때 통화 단위, 세 자리 숫자 단위 띄어쓰기, % 출력 등 다양한 형식에 맞추어 출력할 일이 생기는데, 이를 서식 지정(formatting)이라고 한다.



## 03. 문자열 서식 지정

### ■ % 서식과 format( ) 함수

- 문자열의 서식(format)을 설정할 때, print( ) 함수는 기본적인 출력 형식 외에 % 서식과 format( ) 함수를 구문으로 사용하여 출력 양식을 지정할 수 있다.

코드 6-2 formatting1.py

```
1 print(1, 2, 3)
2 print("a" + " " + "b" + " " + "c")
3 print("%d %d %d" % (1, 2, 3))
4 print("{} {} {}".format("a", "b", "c"))
```

```
1 2 3
a b c
1 2 3
a b c
```

## 03. 문자열 서식 지정

### ■ % 서식과 format( ) 함수 : [코드 6-2] 해석

- 1~2행 : 별도의 서식 지정 없이 그대로 print( ) 함수를 사용하였다.
- 3~4행 : % 서식 지정과 format( ) 함수를 사용하였다. 3~4행의 구문을 사용할 경우 뒤에 있는 숫자와 문자들이 앞의 코드에 대응하여 할당된다. 즉, 3행의 "%d %d %d" %(1, 2, 3)에서 1, 2, 3이 각각 첫 %d부터 차례로 할당된다.
- 4행 : "{} {} {}".format("a", "b", "c")에서 아무것도 적혀 있지 않은 {} 공간에 "a", "b", "c"라는 문자열 형태의 값 3개가 할당되어 출력된다.

## 03. 문자열 서식 지정

### ■ % 서식과 format( ) 함수

- 이런 식으로 서식을 지정하여 출력하면 어떤 장점이 있을까?
- ① 데이터와 출력 형식을 분류할 수 있다. 같은 내용을 여러 번 출력하기 위해 기존 print( )문에 띄어쓰기를 넣어 + 기호로 문자열 형태를 붙여 주는 것보다 시각적으로 훨씬 이해하기 쉽게 코드를 표현할 수 있다.
  - ② 데이터를 형식에 따라 다르게 표현할 수 있다. [코드 6-3]을 보면 문자열 형태인 ('one', 'two') 구문과 정수형인 (1, 2) 구문이 각각 %s와 %d로 다르게 할당되는 것을 확인할 수 있다. 서식 지정 기능은 각 변수의 자료형에 맞게 다른 서식으로 지정한다

코드 6-3 formatting2.py

```
1 print('%s %s' % ('one', 'two'))
2 print('%d %d' % (1, 2))
```

```
one two
1 2
```

## 03. 문자열 서식 지정

### ■ % 서식과 format( ) 함수 : % 서식

- % 서식은 다음과 같은 형태로 출력 양식을 표현하는 기법이다.

'%자료형 % (값)'

- % 서식을 사용한 가장 간단한 표현 형식은 [코드 6-4]와 같다.

코드 6-4 formatting3.py

```
1 print("I eat %d apples." % 3)
2 print("I eat %s apples." % "five")
```

```
I eat 3 apples.
I eat five apples.
```

### 03. 문자열 서식 지정

#### ■ % 서식과 format( ) 함수 : % 서식

서식	설명
%s	문자열(string)
%c	문자 1개(character)
%d	정수(integer)
%f	실수(floating-point)
%o	8진수
%x	16진수
%%	문자 % 자체

[변수의 자료형에 따른 서식]

## 03. 문자열 서식 지정

### ■ % 서식과 format( ) 함수 : % 서식

- %는 1개 이상의 값도 할당할 수 있다. 다음 코드처럼 % 뒤에 괄호를 쓰고, 그 안에 순서대로 값을 입력하면 된다.

```
>>> print("Product: %s, Price per unit: %f." % ("Apple", 5.243))  
Product: Apple, Price per unit: 5.243000.
```

- 직접 값을 넣지 않고 number와 day 같은 변수명을 넣어도 문제없이 실행된다.

**코드 6-5** formatting4.py

```
1 number = 3  
2 day = "three"  
3 print("I ate %d apples. I was sick for %s days." % (number, day))
```

```
I ate 3 apples. I was sick for three days.
```

## 03. 문자열 서식 지정

### ■ % 서식과 format( ) 함수 : format( ) 함수

- **format( ) 함수** : % 서식과 거의 같지만, 문자열 형태가 있는 함수를 사용한다는 차이점이 있다. 문자열 서식은 함수이므로 다음과 같은 형태로 서식을 지정할 수 있다.

```
"{자료형}".format(인수)
```

- 다음 코드는 format( ) 함수를 사용한 가장 기본적인 표현 형태로, 숫자 20이 {0}에 할당되어 출력된다. 기존 % 서식과 비교하면, 자료형을 바로 지정해 주지 않고 순서대로 변수가 할당된다는 장점이 있다.

```
>>> print("I'm {0} years old.".format(20))  
I'm 20 years old.
```

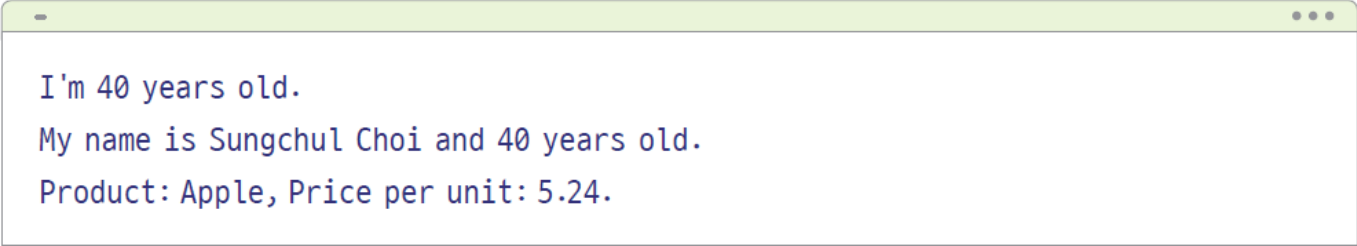
## 03. 문자열 서식 지정

### ■ % 서식과 format( ) 함수 : format( ) 함수

- format( ) 함수는 % 서식처럼 변수의 이름을 사용하거나 변수의 자료형을 따로 지정하여 출력한다.

코드 6-6 formatting5.py

```
1 age = 40; name = 'Sungchul Choi'
2 print("I'm {0} years old.".format(age))
3 print("My name is {0} and {1} years old.".format(name, age))
4 print("Product: {0}, Price per unit: {1:.2f}.".format("Apple", 5.243))
```



```
I'm 40 years old.
My name is Sungchul Choi and 40 years old.
Product: Apple, Price per unit: 5.24.
```

- ➡ 4행의 Price per unit: {1:.2f}는 기존 format( ) 함수의 쓰임과 다르게 .2f라는 구문이 추가되었다. 2는 소수점 둘째 자리까지 출력하라는 뜻이다.



## 03. 문자열 서식 지정

---

### ■ 패딩

- 파이썬의 서식 지정 기능에는 여유 공간을 지정하여 글자 배열을 맞추고 소수점 자릿수를 맞추는 패딩(padding)기능이 있다. % 서식과 `format()` 함수 모두 패딩 기능을 제공한다.

### 03. 문자열 서식 지정

#### ■ 패딩 : % 서식의 패딩

첫 번째 줄의 `print("%10d" % 12)`는 10자리의 공간을 확보하고, 우측 정렬로 12를 출력하라는 명령이다. [우측 정렬 기본](#).

```
>>> print("%10d" % 12)
```

```
12
```

```
>>> print("%-10d" % 12)
```

```
12
```

[좌측정렬](#)을 하기 위해서는 세 번째 줄처럼 - 부호를 붙이면 된다.

우측 정렬과 좌측 정렬의 다른 사례

```
In[10]: print('a=%5d, b=%-5d***'%(10, 20))
```

```
a= 10, b=20
```

```
***
```

우측정렬  
(3칸 공백)

좌측 정렬  
(3칸공백)

### 03. 문자열 서식 지정

#### ■ 패딩 : % 서식의 패딩

```
>>> print("%10.3f" % 5.94343)      # 10자리를 확보하고 소수점 셋째 자리까지 출력
      5.943
>>> print("%10.2f" % 5.94343)      # 10자리를 확보하고 소수점 둘째 자리까지 출력
      5.94
>>> print("%-10.2f" % 5.94343)     좌측 정렬
5.94
```

실수에서도 자릿수와 소수점 자릿수를 지정할 수 있다.

- ➔ 첫 번째 줄의 `print("%10.3f" % 5.94343)`은 10자리의 공간을 확보하고 소수점 셋째 자리까지 출력하라는 뜻이다. 이때 10자리 안에는 소수점이 포함된다. 역시 우측 정렬 기준이며, 좌측 정렬을 하기 위해서는 - 부호를 붙이면 된다.

```
a = 12.34578
b = -12.34578
print('a=%7.2f, b=%-7.2f***' % (a, b))
```

우측 정렬 총 7자리 반올림      부호 포함 좌측 정렬 총 7자리. 반올림.

↓      ↓

➔ a= 12.35, b=-12.35 \*\*\*

## 03. 문자열 서식 지정

### ■ 패딩 : `format()` 함수의 패딩

```
>>> print("{0:>10s}".format("Apple"))
      Apple
>>> print("{0:<10s}".format("Apple"))
Apple
```

- 첫 번째 줄의 `print("{0:>10s}".format("Apple"))`은 10자리의 공간을 확보하고, 우측 정렬(>)로 문자열 'Apple'을 출력하라는 명령이다. 좌측 정렬을 하기 위해서는 '{0:<10s}'처럼 "<" 부호를 사용하면 된다. `s`는 string 형식을 의미한다.

4 <code>print('{0:&lt;10s}***'.format('Hello'))</code> 5 <code>print('{0:&gt;10s}***'.format('Hello'))</code> 6 <code>print('{0:&lt;10d}***'.format(123))</code> 7 <code>print('{0:&lt;10f}***'.format(23.38))</code> 8 <code>print('{0:&gt;10f}***'.format(23.38))</code> 9 <code>print('{0:10.2f}***'.format(-23.38))</code>	➔	Hello*** Hello*** 123*** 23.380000*** 23.380000*** -23.38***
---	---	---

# 확보되는 공간이 나타낼 수의 자릿수보다는 커야 한다. 이 전제가 지켜져야 자릿수와 좌우측 정렬이 올바르게 진행된다.

# 기본적으로 부동소수는 소수 이하 6자리를 출력한다.

## 03. 문자열 서식 지정

### ■ 패딩 : `format( )` 함수의 패딩

```
>>> "{1:10.5f}.".format("Apple", 5.243)
' 5.24300.' Console에서 출력하면 '따옴표'가 출력된다. print()문을 쓰면 없어짐.
>>> "{1:>10.5f}.".format("Apple", 5.243)
' 5.24300.'
>>> "{1:<10.5f}.".format("Apple", 5.243)
'5.24300 .'

```


➡ 실수에서도 자릿수와 소수점 자릿수를 지정할 수 있다.

첫 번째 줄의 `"{1:>10.5f}.".format("Apple", 5.243)`을 입력하면, 10자리의 공간을 확보하고, 소수점 다섯 번째 자리까지 실수를 출력한다. 이때 10자리 안에는 소수점이 포함된다. 역시 우측 정렬 기준이며, 좌측 정렬을 위해서는 `<` 부호를 사용한다.

# 문자열 서식 추가 연습

padding.py

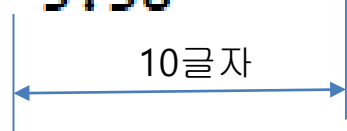
```
print('{0:<10s}***'.format('Hello'))
print('{0:>10s}***'.format('Hello'))
print('{0:<10d}***'.format(123))
print('{0:<10f}***'.format(23.38))
print('{0:>10f}***'.format(23.38))
print('{0:>10.2f}***'.format(-3.3853565))
print('{0:<10.2f}***'.format(-3.3823565))
```



```

Hello***
      Hello***
123***
23.380000***
23.380000***
      -3.39***
-3.38***

```



# 확보되는 공간이 나타낼 수의 자릿수보다는 커야 한다는 전제하에 자릿수와 좌우측 정렬이 올바르게 진행된다.

# 부동소수는 지정하지 않으면 소수 이하 6자리를 출력한다.

## 03. 문자열 서식 지정

### 여기서 잠깐! 네이밍(naming): keyword 지정

- 서식 지정을 활용하여 print() 함수를 출력할 때 한 가지 더 알아야 하는 점은 변수명을 서식에 할당할 수 있는 네이밍이라는 기능이 있다는 것이다. 다음 코드에서 보듯이 기존 번호나 순서대로 자료형에 대응하는 것이 아닌, 'name'이나 'price'처럼 특정 변수명을 사용하여 출력값에 할당할 수 있다. 특히 한 번에 출력해야 하는 변수가 많을 때, 개발자 입장에서 변수의 순서를 헷갈리지 않고 사용할 수 있다는 장점이 있다.

```
>>> print("Product: %(name)5s, Price per unit: %(price)5.5f." %  
{"name": "Apple", "price": 5.243})  
Product: Apple, Price per unit: 5.24300.  
>>> print("Product: {name:>5s}, Price per unit: {price:5.5f}.".format(name="A  
pple", price=5.243))  
Product: Apple, Price per unit: 5.24300.
```

# OS module

```
In[11]: import os
In[12]: os.getcwd()
Out[12]: 'D:\\\\Work\\\\@@PythonProgramming\\\\LectureMaterials'
In[13]: os.chdir('..')
In[14]: os.getcwd()
Out[14]: 'D:\\\\Work\\\\@@PythonProgramming'
In[15]: a = os.getcwd()
In[16]: os.chdir('C:\\Windows')
In[17]: os.getcwd()
Out[17]: 'C:\\\\Windows'
In[18]: os.chdir(a)
In[19]: os.getcwd()
Out[19]: 'D:\\\\Work\\\\@@PythonProgramming'
```

- `cetcwd`: 현재의 폴더를 출력
- `chdir('..')`: 한 단계 위쪽 폴더로 이동
- `chdir('c:\\windows')`: `c:\\windows` 폴더로 이동



# OS module

코드 페이지는 숫자, 구두점, 기타 그림 문자 등을 포함하는 문자 집합이다.  
각 언어와 로케일마다 다른 코드 페이지를 사용한다.

대표적인 코드 페이지 값

949 : 한국어 (EUC-KR), 65001 : 유니코드 (UTF-8)

```
In[34]: os.system('chcp')
```

```
0 949 :  J
```

*현재의 코드 페이지를 출력한다, 현재 949*

```
Out[34]: 0
```

```
In[35]: os.system('chcp 65001')
```

*파이참(실행창, 콘솔창)의 한글 출력을 지원*

```
Active code page: 65001
```

```
Out[35]: 0
```

```
In[36]: os.system('dir jkasd')
```

*파일 jkasd의 기본 내용을 출력하라*

```
Volume in drive D is Work(Segate_3T)
```

```
Volume Serial Number is D489-981F
```

```
Directory of D:\Work\@@PythonProgramming\LectureMaterials
```

```
Out[36]: 1
```

```
File Not Found
```

```
In[37]: os.system('dir jkasd > screen.txt')
```

*화면에 출력한 내용을 파일로 저장하라,*

```
File Not Found
```

```
Out[37]: 1
```

```
In[38]: os.system('type screen.txt')
```

*문서 파일을 출력하라,*

```
Volume in drive D is Work(Segate_3T)
```

```
Volume Serial Number is D489-981F
```

```
Directory of D:\Work\@@PythonProgramming\LectureMaterials
```

```
Out[38]: 0
```

# File encoding

```
In[5]: import sys
In[6]: sys.getdefaultencoding()
Out[6]: 'utf-8'
```

시스템이 사용중인 디폴트 엔코딩 설정을 확인하는 방법

```
f = open('test.txt', mode='wt', encoding='utf-8')
f.write("Hello, World\n");
f.write("안녕하세요?")
f.close()
```

텍스트 파일에 두 라인의 텍스트를 쓰는 코드

# 1차 레포트 공고(1/2)

- 특정 폴더(변수 a) 안에 있는 특정 파일(예: pptx, py 파일 등)의 개수(수량) 및 그 확장자를 갖는 모든 파일의 용량(바이트)를 출력하는 프로그램을 작성하시오.
- 점검 방법은 향후 파일이 담긴 폴더를 제공하고 그곳에 담긴 내용을 분석하여 출력하는 내용을 판단하여 통과 여부를 결정할 것임.
- 프로그램은 다음과 같이 지정한 위치의 파일 정보를 출력해야 함.

확장자의 개수와 각 확장자에 따른 파일의 수를 출력하시오.

확장자: 개수: 파일크기의 합:

py:24: 1000

exe: 2: 2002

txt: 3: 10

<abdc>

<폴더1>

총 확장자의 수=3, 폴더의 수=2, 총 파일의 수=29, 총 용량=3012

# 1차 레포트 공고(2/2)

## ■ 제출요령 및 점검 방법은 차후 고지함

### ■ 1. 제출 기한: 다음 주 수업 시간.

- 상황에 따라 내용이 추가되어 제출 기한이 연장될 수도 있음

### ■ 2. 제출 내용

- 1) 레포트: PDF, 프로그램 전체 흐름 설명, 주요 루틴 설명, 스스로 정한 사례로 실험한 결과물이 반영되어 있어야 함.
- 및 평가, 맺음말
- 2) 소스프로그램, 주석문 명기
- 3) 발표 동영상: 3분 이내.

## ■ 제출방법

- 다음 시간에 고지. 각자 제시한 방법에 따라 제출할 준비를 해 놓기 바람.
- 사례: 구글 클래스 등...

# Thank You !