python 데이터의 비밀

☑ 복습	☐
≡ 개념	
☑ 블로그	☐
≡ 페이지	

Aliasing

```
x = 5
y = x
y = 3
print(x)
print(y)
#--==>>
'''
5
3
'''

a = [2, 3, 5, 7, 11]
b = a       # a리스트에 b라는 이름표를 달아줌
b[2] = 4    # 리스트b의 2번째 자리를 4로 변경 하지만 이 리스트는 a이기도 하고 b이기도 하다!
print(a)
print(b)
#--==>> 그래서 둘다 같은 리스트 출력 여기에서 b는 a의 가명(alias)라고 한다!
'''
[2, 3, 4, 7, 11]
[2, 3, 4, 7, 11]
'''
```

그렇다면 b를 바꾸면서 a의 값을 그대로 유지하는 방법은?

```
a = [2, 3, 5, 7, 11]        # 이 리스트에 a의 이름표가 달림
b = list(a)
# list함수 때문에 새로운 리스트가 그대로 복사되어 새로운 리스트를 만든다
# 여기서 list 함수의 역할은 새로운 리스트를 복사해주는 것이다.
b[2] = 4
print(a)
print(b)
#--==>> b리스트만 바뀐다.
'''
[2, 3, 5, 7, 11]
[2, 3, 4, 7, 11]
'''
```

3 리스트와 문자열

alphabet_list = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']print(alphabet_list[0])print(alphabet_list[1])print(alphabet_list[4])print(alphabet_list[-1])# 문자열과 리스트는 비슷하다/ 이것을 문자열에서도 똑같이 인덱싱 할 수 있다

```
alphabet_list = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

print(alphabet_list[0])
print(alphabet_list[1])
print(alphabet_list[4])
print(alphabet_list[-1])
#--==>>
'''
a
b
e
j
'''


# 문자열과 리스트는 비슷하다
# 이러한 방법으로 문자열에서도 똑같이 인덱싱할 수 있다.
alphabet_string = 'abcdefghij'

print(alphabet_string[0])
print(alphabet_string[1])
print(alphabet_string[4])
print(alphabet_string[-1])
#--==>>
'''
a
b
e
j
'''
```

리스트 슬라이싱도 보겠다

```
alphabet_list = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']

print(alphabet_list[0:5])
print(alphabet_list[4:])
print(alphabet_list[:4])
#--==>>
'''
['a', 'b', 'c', 'd', 'e']
['e', 'f', 'g', 'h', 'i', 'j']
['a', 'b', 'c', 'd']
```

```
'''


# 문자열에서도 똑같이 동작한다.(문자열 슬라이싱)
alphabet_string = 'abcdefghij'

print(alphabet_string[0:5])
print(alphabet_string[4:])
print(alphabet_string[:4])
#--==>>
'''
abcde
efghij
abcd
'''
# 리스트와 문자열이 구조적으로 비슷하기 때문에 이 모든게 가능하다.
```

```
str_1 = "hello"
str_2 = "world"

str_3 = str_1 + str_2
print(str_3)
#--==>> helloworld

# 문자열처럼 리스트를 연결하는 것도 가능
list_1 = [1, 2, 3, 4]
list_2 = [5, 6, 7, 8]
list_3 = list_1 + list_2
print(list_3)
#--==>> [1, 2, 3, 4, 5, 6, 7, 8]
```

```
my_list = [2, 3, 5, 7, 11]
print(len(my_list))
#--==>> 5

# len 함수를 문자열에서도 가능
my_string = "hello world!"
print(len(my_string))
#--==>> 12 (띄어쓰기도 포함)
```

```
numbers = [1, 2, 3, 4]
numbers[0] = 5
print(numbers)
#--==>> [5, 2, 3, 4]        인덱스 0의 값을 5로 변경

name = "Lee"
```

```
name[0] = "C"
print(name)
#-==>> 오류 발생(TypeError: 'str' object does not support item assignment)
# 문자열은 리스트와 달리 수정이 불가능하다
```

리스트와 문자열은 굉장히 비슷합니다. 리스트가 어떤 자료형들의 나열이라면, 문자열은 문자들의 나열이라고 할 수 있겠죠. 지금부터 파이썬에서 리스트와 문자열이 어떻게 같고 어떻게 다른지 알아봅시다.

인덱싱 (Indexing)

두 자료형은 공통적으로 인덱싱이 가능합니다.

```
# 알파벳 리스트의 인덱싱
alphabets_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']
print(alphabets_list[0])
print(alphabets_list[1])
print(alphabets_list[4])
print(alphabets_list[-1])

# 알파벳 문자열의 인덱싱
alphabets_string = 'ABCDEFGHIJ'
print(alphabets_string[0])
print(alphabets_string[1])
print(alphabets_string[4])
print(alphabets_string[-1])

#-==>>
'''
A
B
E
J
A
B
E
J
'''
```

for 반복문

두 자료형은 공통적으로 인덱싱이 가능합니다. 따라서 for 반복문에도 활용할 수 있습니다.

```
# 알파벳 리스트의 반복문
alphabets_list = ['C', 'O', 'D', 'E', 'I', 'T']
```

```
for alphabet in alphabets_list:
    print(alphabet)

# 알파벳 문자열의 반복문
alphabets_string = 'CODEIT'

for alphabet in alphabets_string:
    print(alphabet)
#--==>>
'''
C
O
D
E
I
T
C
O
D
E
I
T
'''
```

슬라이싱 (Slicing)

두 자료형은 공통적으로 슬라이싱이 가능합니다.

```
# 알파벳 리스트의 슬라이싱
alphabets_list = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J']

print(alphabets_list[0:5])
print(alphabets_list[4:])
print(alphabets_list[:4])

# 알파벳 문자열의 슬라이싱
alphabets_string = 'ABCDEFGHIJ'

print(alphabets_string[0:5])
print(alphabets_string[4:])
print(alphabets_string[:4])
#--==>>
'''
['A', 'B', 'C', 'D', 'E']
['E', 'F', 'G', 'H', 'I', 'J']
['A', 'B', 'C', 'D']
ABCDE
EFGHIJ
ABCD
'''
```

덧셈 연산

두 자료형에게 모두 덧셈은 "연결"하는 연산입니다.

```
# 리스트의 덧셈 연산
list1 = [1, 2, 3, 4]
list2 = [5, 6, 7, 8]
list3 = list1 + list2
print(list3)

# 문자열의 덧셈 연산
string1 = '1234'
string2 = '5678'
string3 = string1 + string2

print(string3)
#--==>>
'''
[1, 2, 3, 4, 5, 6, 7, 8]
12345678
'''
```

len 함수

두 자료형은 모두 길이를 재는 **len** 함수를 쓸 수 있습니다.

```
# 리스트의 길이 재기
print(len(['H', 'E', 'L', 'L', 'O']))

# 문자열의 길이 재기
print(len("Hello, world!"))
#--==>>
'''
5
13
'''
```

Mutable (수정 가능) vs. Immutable (수정 불가능)

하지만 차이점이 있습니다. 리스트는 데이터를 바꿀 수 있지만, 문자열은 데이터를 바꿀 수 없다는 것입니다. 리스트와 같이 수정 가능한 자료형을 'mutable'한 자료형이라고 부르고, 문자열과 같이 수정 불가능한 자료형을 'immutable'한 자료형이라고 부릅니다. 숫자, 불린, 문자열은 모두 immutable한 자료형입니다.

```
# 리스트 데이터 바꾸기
numbers = [1, 2, 3, 4]
numbers[0] = 5

print(numbers)
#--==>> [5, 2, 3, 4]
```

리스트 **numbers**의 인덱스 **0**에 **5**를 새롭게 지정해주었습니다. **[5, 2, 3, 4]**가 출력되었습니다. 이처럼 리스트는 데이터의 생성, 삭제, 수정이 가능합니다.

```
# 문자열 데이터 바꾸기
name = "codeit"
name[0] = "C"

print(name)
#--==>>
'''
Traceback (most recent call last):
    File "untitled.py", line 3, in <module>
        name[0] = "C"
TypeError: 'str' object does not support item assignment
'''
```

문자열 **name**의 인덱스 **0** 에 **"C"**를 새롭게 지정해주었더니 오류가 나왔습니다. **TypeError: 'str' object does not support item assignment**는 문자열은 변형이 불가능하다는 메시지입니다. 이처럼 문자열은 리스트와 달리 데이터의 생성, 삭제, 수정이 불가능합니다.

문제

함수 sum_digit은 파라미터로 정수형 num을 받고, num의 각 자릿수를 더한 값을 리턴합니다.

예를 들어서 12의 각 자릿수는 1, 2이니까 sum_digit(12)는 3(1 + 2)을 리턴합니다.

마찬가지로 486의 각 자릿수는 4, 8, 6이니까 sum_digit(486)은 18(4 + 8 + 6)을 리턴하는 거죠.

여러분이 해야 할 일은 두 가지입니다.

sum_digit 함수를 작성한다.

sum_digit(1)부터 sum_digit(1000)까지의 합을 구해서 출력한다.

실행 결과

```
13501
```

답

```
# 자리수 합 리턴
def sum_digit(num):
    total = 0
    str_num = str(num)

    for digit in str_num:
        total += int(digit)

    return total

# sum_digit(1)부터 sum_digit(1000)까지의 합 구하기
digit_total = 0
for i in range(1, 1001):
    digit_total += sum_digit(i)

print(digit_total)
```

문제

주민등록번호 **YYMMDD-abcdefg**는 총 열세 자리인데요.

앞의 여섯 자리 **YYMMDD**는 생년월일을 의미합니다.

- **YY** → 연
- **MM** → 월
- **DD** → 일

뒤의 일곱 자리 **abcdefg**는 살짝 복잡합니다.

- **a** → 성별
- **bc** → 출생등록지에 해당하는 지방자치단체의 고유번호
- **defg** → 임의의 번호

보시다시피 많은 부분은 특정 규칙대로 정해져 있는데요. 여러분에 대한 몇 가지 정보만 알면, 마지막 네 개 숫자 **defg**를 제외한 앞의 아홉 자리는 쉽게 알 수 있다는 거죠.

그래서 저희는 주민등록번호의 마지막 네 자리 **defg**만 가려 주는 보안 프로그램을 만들려고 합니다.

mask_security_number라는 함수를 정의하려고 하는데요. 이 함수는 파라미터로 문자열 **security_number**를 받고, **security_number**의 마지막 네 글자를 **'*'**로 대체한 새 문자열을 리턴합니다.

참고로 파라미터 **security_number**에는 작대기 기호(**-**)가 포함될 수도 있고, 포함되지 않을 수도 있는데요. 작대기 포함 여부와 상관 없이, 마지막 네 글자가 **'*'**로 대체되어야 합니다!

실행 결과

```
880720-123****
880720123****
930124-765****
930124765****
761214-235****
761214235****
```

과제 해설

접근법 #1

문자열은 수정이 불가능합니다. 하지만 문자열과 유사한 리스트는 수정이 가능하죠? 그러면 문자열 **security_number**를 리스트로 변환한 후, 마지막 네 원소를 **'*'**로 바꿔 주면 됩니다. 그리고 나서 그 리스트를 다시 하나의 문자열로 합치면 되겠죠?

코드로 봅시다.

```
def mask_security_number(security_number):
    # security_number를 리스트로 변환
    num_list = []
    for i in range(len(security_number)):
        num_list.append(security_number[i])
```

문자열을 반복문을 쓰지 않고 한번에 리스트로 바꾸고 싶으면 곧바로 형 변환을 쓸 수도 있습니다.

```
def mask_security_number(security_number):
    # security_number를 리스트로 변환
    num_list = list(security_number)
```

이제 마지막 네 요소, 즉 인덱스 len(num_list) - 4부터 인덱스 len(num_list) - 1의 값들을 *로 바꿔주면 됩니다.

```
def mask_security_number(security_number):
    # security_number를 리스트로 변환
    num_list = list(security_number)

    # 마지막 네 값을 *로 대체
    for i in range(len(num_list) - 4, len(num_list)):
        num_list[i] = "*"
```

마지막으로 이 리스트를 이제 다시 문자열로 만들어서 리턴시켜 주어야합니다. 리스트의 각 요소를 하나씩 빈 문자열을 시작으로 **연결**해주면 기존 문자열로 만들 수 있을 것입니다. 그럼 이 연결을 어떻게 해줄 수 있을까요? **리스트와 문자열 정리**에서 배운 **덧셈 연산**을 이용하면 됩니다.

```
def mask_security_number(security_number):
    # security_number를 리스트로 변환
    num_list = list(security_number)

    # 마지막 네 값을 *로 대체
    for i in range(len(num_list) - 4, len(num_list)):
        num_list[i] = "*"

    # 리스트를 문자열로 복구
    total_str = ""
    for i in range(len(num_list)):
        total_str += num_list[i]

    return total_str
```

접근법 #2

사실 리스트를 문자열로 복구하는 코드는 **join()** 이라는 메소드로 한번에 할 수 있습니다. 이 메소드는 문자열로 이루어진 리스트를 **구분자**로 결합하여 하나의 문자열로 만들어 줍니다. 아래 코드로 확인해보겠습니다.

```
units = ["cm", "m", "yard"]
units_to_string = ', '.join(units)

print(type(units_to_string))
print(units_to_string)
#--==>>
'''
<class 'str'>
cm, m, yard
'''
```

join() 메소드는 **str.join(list)** 형태로 쓰게 됩니다. **str**은 리스트 요소들을 결합할 때 사용될 **구분자**입니다. 구분자는 **문자열**이어야 합니다. 그리고 **list** 는 각 요소가 문자열인 리스트를 의미합니다. 그래서 위 코드의 결과로 units 리스트의 각 요소들이 **,**와 **공백**으로 결합된 하나의 문자열로 출력되는 것입니다.

이 과제에서는 각 숫자들을 **공백없이** 결합을 해야합니다. 그럼 어떻게 작성할 수 있을까요? 잠시 생각해보시고 아래 코드를 확인해보세요.

```
def mask_security_number(security_number):
    num_list = list(security_number)

    # 마지막 네 값을 *로 대체
    for i in range(len(num_list) - 4, len(num_list)):
        num_list[i] = "*"

    # 리스트를 문자열로 복구해서 반환
    return "".join(num_list)
```

접근법 #3

그런데 더 쉬운 방법이 있습니다. 문자열 슬라이싱을 이용하는 건데요.

security_number의 마지막 네 자리만 제외해서 슬라이싱을 하고, 문자열 **"****"**과 연결하면 끝입니다!

모범 답안

```
def mask_security_number(security_number):
    return security_number[:-4] + "****"

print(mask_security_number("880720-1234567"))
print(mask_security_number("8807201234567"))
print(mask_security_number("930124-7654321"))
print(mask_security_number("9301247654321"))
print(mask_security_number("761214-2357111"))
print(mask_security_number("7612142357111"))
```

문제

"토마토"나 **"기러기"**처럼 거꾸로 읽어도 똑같은 단어를 '팰린드롬(palindrome)'이라고 부릅니다.

팰린드롬 여부를 확인하는 함수 **is_palindrome**을 작성하려고 하는데요. **is_palindrome**은 파라미터 **word**가 팰린드롬이면 **True**를 리턴하고 팰린드롬이 아니면 **False**를 리턴합니다.

예를 들어서 **"racecar"**과 **"토마토"**는 거꾸로 읽어도 똑같기 때문에 **True**가 출력되어야 합니다. 그리고 **"hello"**는 거꾸로 읽으면 **"olleh"**가 되기 때문에 **False**가 나와야 하는 거죠.

실행 결과

```
# 테스트
print(is_palindrome("racecar"))
print(is_palindrome("stars"))
print(is_palindrome("토마토"))
print(is_palindrome("kayak"))
print(is_palindrome("hello"))
#--==>>
'''
True
False
True
True
False
'''
```

해설

문자열의 첫 번째 원소와 마지막 원소를 비교해서 일치하는지 확인해야 합니다. 그 다음 문자열의 두 번째 원소와 끝에서 두 번째 원소를 비교해서 일치하는지 확인해야겠죠.

문자열 **word**의 첫 번째 원소의 인덱스는 **0**이고, 마지막 원소의 인덱스는 **len(word) - 1**입니다. 문자열 **word**의 두 번째 원소의 인덱스는 **1**이고, 끝에서 두 번째 원소의 인덱스는 **len(word) - 2**입니다.

이걸 어떻게 일반화할 수 있을까요?

i를 **0**부터 **1**씩 늘린다고 가정했을 때, 인덱스 **i**에 있는 값과 인덱스 **len(word) - i - 1**에 있는 값을 비교하면 됩니다!

참고로 **i**를 **0**부터 **len(word) - 1**까지 반복할 필요는 없습니다. 어차피 반대쪽과 비교하는 것이기 때문에 **i**를 **len(word) // 2**까지만 반복해도 이미 모든 확인은 끝나는 거죠!

모범 답안

```
def is_palindrome(word):
    for left in range(0, len(word) // 2):
        # 한 쌍이라도 일치하지 않으면 바로 False를 리턴하고 함수를 끝냄
        right = len(word) - left - 1
        if word[left] != word[right]:
            return False

    # for문에서 나왔다면 모든 쌍이 일치
    return True

print(is_palindrome("racecar"))
print(is_palindrome("stars"))
print(is_palindrome("토마토"))
```

```
print(is_palindrome("kayak"))
print(is_palindrome("hello"))
#--==>>
'''
True
False
True
True
False
'''
```