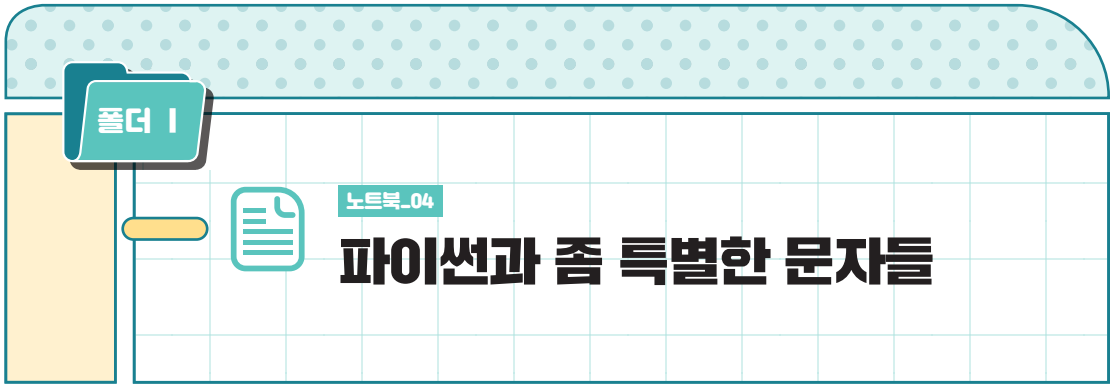
46p In [1] 코드. 문자열 따옴표 색 오류



이전 노트북에서 알파벳과 한글 그리고 숫자까지는 이야기하는 데 성공했습니다. 그런데 이것만으로 모든 의사표현을 다 할 수 있는 것은 아닙니다.

가령 키보드에는 알파벳/한글이나 숫자, 기호 같은 일반 키뿐 아니라, 누르면 어떤 작업을 수행하는 특수 키들이 있습니다. [Backspace], [Enter] 키 등이 대표적이지요. 그런가 하면 공간의 한계로 키보드에 추가되지 못한 온갖 특수 문자들도 있습니다. 또한 우리는 한자나 일본 문자, 그리스 문자를 입력하고 싶을지도 모릅니다!

그런 다문자 사용자의 꿈을 위해, 파이썬의 해결책을 만나 보겠습니다.

이스케이프 문자: \

문자나 숫자 입력이 아닌 '기능'을 입력하는 키들을 구현하기 위해, 프로그램 코드에서는 \ 기호로 시작하는 문자 조합을 사용합니다. 예를 들어 앞에서 이미 확인했지만, '줄바꾸기' 기능은 \n 문자로써 표현되고, [Enter] 키 입력을 대체합니다.

```
In [1]: print('많이 보고 싶겠지만\n조금만 참자')      # \n = 줄바꾸기 문자

많이 보고 싶겠지만
조금만 참자
```

76p In [1] 코드. True 색상

```
In [1]:
type(1)          # 정수형(int)
type(1.0)        # 실수형(float)
type(2+3j)       # 복소수형(complex)
type('abc')      # 문자열(str)
type(0x1234)     # 정수형(int)
type(True);      # 불형(bool)
```

코드 맨 마지막 ;은 입력해도 되고, 하지 않아도 무방합니다.

주피터 노트북에서 마지막의 ;은 실행 시 최종 출력을 생략하기 위해 쓰입니다. 원래 Out 블록이 자동으로 출력되는 코드라도, 반환값이 없도록 해줍니다.

표 파이썬의 기초 자료형 예시

자료형	예시
정수형(int)	1, 2, -1, -2, 0과 같이, 소수점이 없는 수치 자료
실수형(float)	1.2, 2.5, 1.4E12와 같이, 소수점이 있거나 지수부가 있는 수치 자료
복소수형(complex)	2+3j, 1+2J 와 같이, 실수부와 허수부가 있는 수치 자료
문자열(str)	'안녕하세요', "144"와 같이, 따옴표로 둘러싸인 자료
불형(bool)	True, False와 같이, 참 또는 거짓을 나타내는 자료

수치 자료형

정수형 상수

정수형 상수는 10진, 8진, 16진, 2진 상수로 표현할 수 있습니다.

```
In [2]:
123              # 123,  10진 상수
0o123            # 83,   8진 상수(octal)
0x123            # 291,  16진 상수(hexa)
0b10110101;      # 181,  2진 상수(binary)
```

2진수, 8진수, 16진수 => 모두 이진수!

이진수 세 자리 = 8진수

이진수 네 자리 = 16진수

정수형 자료에서는 수치 표현 범위가 제한되지 않습니다. 다음과 같은 30자리의 정수도 거뜬히 표현합니다.

리스트의 값(요소)은 언제든지 변경 가능(mutable)합니다. `L[n] = 변경값` 구문의 오른쪽에 변경할 값을 넣고, 몇 번째 요소를 변경할 것인지 n에 써 주면 됩니다.

> 되짚어보기
>
> 리스트의 요소를 뒤에서부터 선택하려면 음수를 사용합니다.
>
> 예: L 리스트에서 맨 뒤에 있는 5를 출력하고 싶다면 `L[-1]`로 작성

```
In [5]:
L[0] = 100      # 첫 번째 값 변경(1 →100)
L

[100, 2, 3, 4, 5]
```

요소의 삭제는 del을 이용합니다.

```
In [6]:
del L[0]      # 첫 번째 요소(100) 제거
L

[2, 3, 4, 5]
```

슬라이싱을 이용해서 구간 요소를 삭제할 수도 있습니다.

```
In [7]:
del L[2:]      # [2, 3, 4, 5]에서 2번 항목 이후 전부 제거
L

[2, 3]
```

리스트에서 메서드 사용하기

리스트 역시, 기초 자료형처럼 메서드를 활용할 수 있습니다. 사용 방식은 다음과 같습니다.

```
리스트이름.메서드(요소)
```

리스트 마지막에 요소를 추가하려면 append() 메서드를 사용합니다.

```
Input In [20]
  t2 = (1 2)
       ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

한편, 튜플은 다음과 같이 리스트에 담긴 형태로도 많이 사용됩니다.

```
In [21]:
T = [('one', 1), ('two', 2), ('three', 3)]
```

튜플과 리스트의 가장 큰 차이는 '변경 가능성'에 있습니다. 튜플은 값을 변경할 수 없습니다(immutable). 변경하려 하면, `TypeError`가 발생합니다.

```
In [22]:
t[0] = 100      # 앞에서 정의한 튜플의 '1' 요소를 '100'으로 변경 시도

---------------------------------------------------------------
TypeError                 Traceback (most recent call last)
Input In [22], in <cell line: 1>()
      1 t = ()
      2 t = (1, 2, 3)
----> 3 t[0] = 100

TypeError: 'tuple' object does not support item assignment
```

○ 복합 자료형을 구분하는 속성 ○
시퀀스형 여부 / 변경 가능 여부 / 중복 허용 여부 / 매핑 여부

튜플은 포함된 요소나 값을 변경할 수 없으므로, 변경이나 추가, 삭제 등을 시도하면 오류가 발생합니다.

따라서 튜플은 변경하지 않을 데이터를 보관하는 용도로 주로 사용되며, 검색에 관련된 메서드 몇 개만 지원합니다. 요소를 세는 `count()` 메서드나,

```
In [23]:
t = (1, 2, 3)      # 튜플 t 재정의
t.count(3)         # 튜플 안에 있는 요소 3의 개수는?

1
```

요소 위치를 찾아 주는 `index()` 메서드가 그 예입니다.

리스트 `[1, 2, 3, 4, 5, 3, 4, 5]`를 `set()` 메서드에 넣었더니, `{1, 2, 3, 4, 5}`라는 집합이 되었습니다. 이렇게 된 것은, 집합은 중복 요소를 허용하지 않기 때문입니다.

그렇다면 이건 어떨까요? 문자열 `'HELLO'`의 문자들을 이용해 요소 순서가 다른 두 집합을 준비해서, `==` 연산자로 값을 비교해 보겠습니다.

```
In [23]: {'E', 'H', 'L', 'O'} == {'H', 'E', 'L', 'O'}

         True
```

같은 값이군요! 그렇습니다. 집합에는 '순서'의 개념도 없습니다. 시퀀스 자료형과 달리, 요소들의 위치가 달라져도 동일 요소를 똑같이 갖고 있기만 하면 동일 집합으로 취급됩니다.

집합 메서드와 집합 고유의 연산

리스트나 사전과 마찬가지로, 집합 자료형에서도 요소를 추가하거나 수정, 제거하는 변경 작업이 가능합니다. 다음 집합 A로 테스트해 보겠습니다.

```
In [24]: A = {1, 2, 3, 4, 5}
```

`add()` 메서드로 집합 A에 원소 10을 추가합니다.

```
In [25]: A.add(10)
         A

         {1, 2, 3, 4, 5, 10}
```

`discard()` 메서드로 집합 A에서 원소 1을 버립니다.

```
In [26]: A.discard(1)
         A
```

215p In [11] 코드. 오자 수정 : LookupErrorError → LookupError

```
In [11]:
class BadUserNameError(LookupError):
    pass

class BadPasswordError(LookupError):
    pass
```

이것은 클래스입니다. 파이썬에서 중요한 개념이지만, 사용자 예외를 만드는 데 있어서 클래스에 대한 깊은 이해는 당장 필요 없으니, 자세한 설명은 나중으로 미루겠습니다.

그러면 `LookupError` 아래에 `BadUserNameError`, `BadPasswordError` 두 개의 예외 클래스가 생성됩니다. 이로써 사용자 예외를 쉽게 정의할 수 있습니다.

이렇게 정의된 예외는 앞서 살펴본 `raise` 문을 이용해 발생시킬 수 있고, 또 `except` 문을 이용해 오류를 잡아낼 수도 있습니다.

```
In [12]:
try:
    raise BadUserNameError
except BadUserNameError:
    print('...')

...
```

실행과제

'a.txt' 파일을 여는 open('a.txt') 작업을 수행하려고 합니다. FileNotFoundError 오류가 발생하는 경우, open('b.txt')를 실행하도록 try...except 구조를 이용하여 구현해 보세요.

여러분이 정의하는 사용자 정의 예외를 만들고, raise와 except 문을 통해서 예외를 처리해 보세요.

acc의 초깃값 1에다 2를 더해서, 3이 acc 변수에 저장되었습니다. 같은 방식으로 5까지 덧셈을 해 나가면, 1부터 5까지의 누적값을 계산할 수 있습니다.

In [9]:
```
acc = 0
acc = acc + 1    #0+1
acc = acc + 2    #1+2
acc = acc + 3    #3+3
acc = acc + 4    #6+4
acc = acc + 5    #10+5

acc    #acc 값은?
```

```
15
```

이 코드를 for 문을 이용해서 다시 표현해 보겠습니다.

In [10]:
```
acc = 0
for i in range(1, 6):    #i를 1부터 5까지 반복
    acc = acc + i
acc
```

```
15
```

이제 더 큰 범위 수의 합계도 계산할 수 있습니다. 다음은 1부터 1000까지의 합연산입니다.

In [11]:
```
acc = 0
for i in range(1, 1000+1):    #i를 1부터 1000까지 반복
    acc = acc + i
acc
```

```
500500
```

> '?는 단순히 ?를 ?해서는 감염이 되지 않는다. 이에 따라 ?으로 설계된 ?가 필요하다.'

`m.groups()`는 두 그룹의 매칭된 문자열을 튜플에 담아 반환해 줍니다. 한자라면 첫째 항목에, 알파벳이라면 둘째 항목에 매칭 결과가 담겨 있습니다. `sub()` 함수인 경우 매칭되지 않으면 `None`으로 채워지네요.

한 번에 전부 매칭되어 편리하군요. 이 함수 `f`를 이용하면 한자와 영어의 태그를 별도로 붙일 수 있겠습니다.

In [37]:
```
def f(m):
    t = m.group()      # 매칭된 문자열 튜플 t에 대하여
    if t[0]:           # 첫 번째 항목(인덱스 0)이 존재하면 한자
        return '<hanja>{}</hanja>'.format(t[0])
    else:              # 첫 번째 항목이 존재하지 않으면(=None) 영어
        return '<eng>{}</eng>'.format(t[1])
    return t

s = 'HIV는 단순히 cDNA를 transfection해서는 감염이 되지 않는다. 이에 따라 逆相遺傳學的으로 설계된 vector가 필요하다.'
re.sub('([一-龥]+)|([a-zA-Z]+)', f, s)
```

> '<eng>HIV</eng>는 단순히 <eng>cDNA</eng>를 <eng>transfection</eng>해서는 감염이 되지 않는다. 이에 따라 <hanja>逆相遺傳學的</hanja>으로 설계된 <eng>vector</eng>가 필요하다.'

4. 단어 경계 찾기

이번에는 동일한 문자로 구성된 문자열의 경우를 살펴보고자 합니다. 한글로 된 다음 문장에서 사람 이름을 구별하고, 거기에 `<person></person>`이라는 태그를 붙여 보겠습니다.

```
'야곱과 에서는 집에서 나와 들로 나갔습니다.'
⇒ '<person>야곱</person>과 <person>에서</person>는 집에서 나와 들로 나갔습니다.'
```

따라서 행 중심으로 꺼내고 싶다면 `transpose()` 메서드로 `DataFrame`의 행과 열을 맞바꾼 후에,

In [13]:
```
df.transpose()
```

	0	1
name	홍길동	홍길순
phone	010-1234-5678	010-7777-8888
total_price	38000	23000
due_date	2021.05.21	2021.05.21
address	경기도 파주시	경기도 부천시

사전으로 변환할 수 있습니다.

In [14]:
```
d = df.transpose().to_dict()
d
```

```
{0: {'name': '홍길동',
  'phone': '010-1234-5678',
  'total_price': 38000,
  'due_date': '2021.05.21',
  'address': '경기도 파주시'},
 1: {'name': '홍길순',
  'phone': '010-7777-8888',
  'total_price': 23000,
  'due_date': '2021.05.21',
  'address': '경기도 부천시'}}
```

사전으로 표현된 고객의 정보를 리스트에 담고 싶다면, 앞에서 추출된 사전에서 값의 목록만 취하면 됩니다.

In [15]:
```
list(df.transpose().to_dict().values())
```